

版权注意事项：

- 1、书籍版权归作者和出版社所有
- 2、本PDF仅限用于个人获取知识，进行私底下的知识交流
- 3、PDF获得者不得在互联网上以任何目的进行传播
- 4、如觉得书籍内容很赞，请购买正版实体书，支持作者
- 5、请于下载PDF后24小时内删除本PDF。



范例文件下载



Hadoop+Spark

大数据巨量分析与机器学习

整合开发实战

林大贵 著

- 大数据技术为金融财务、营销分析、商业趋势预测带来全新变革
- 详实的安装设置与程序解析，降低学习门槛
- 可单机运行/实机/虚拟机建立多台运算集群
- 提供大量实际案例详解与程序代码范例



清华大学出版社

内容简介

随着大数据的兴起，企业、政府、科研机构等纷纷投入巨资进行大数据的采集、存储、分析和应用。本书以Hadoop和Spark为核心，详细介绍了大数据的存储、计算、分析和应用。本书适合从事大数据开发、运维、管理和应用的人员阅读，也可作为高等院校计算机专业及相关专业的教材。

本书共分10章。第1章介绍大数据的背景和概念；第2章介绍Hadoop的架构和原理；第3章介绍Hadoop的分布式文件系统HDFS；第4章介绍Hadoop的分布式计算框架MapReduce；第5章介绍Hadoop的分布式数据库HBase；第6章介绍Hadoop的分布式搜索引擎Elasticsearch；第7章介绍Hadoop的分布式图计算框架GraphX；第8章介绍Hadoop的分布式机器学习框架Spark；第9章介绍Hadoop的分布式数据挖掘框架Mahout；第10章介绍Hadoop的分布式数据可视化框架Tableau。

Hadoop+Spark

大数据巨量分析与机器学习

整合开发实战

林大贵 著

清华大学出版社

北京

内 容 简 介

本书从浅显易懂的“大数据和机器学习”原理介绍和说明入手,讲述大数据和机器学习的基本概念,如:分类、分析、训练、建模、预测、机器学习(推荐引擎)、机器学习(二元分类)、机器学习(多元分类)、机器学习(回归分析)和数据可视化应用。为降低读者学习大数据技术的门槛,书中提供了丰富的上机实践操作和范例程序详解,展示了如何在单台 Windows 系统上通过 Virtual Box 虚拟机安装多台 Linux 虚拟机,如何建立 Hadoop 集群,再建立 Spark 开发环境。书中介绍搭建的上机实践平台并不限于单台实体计算机。对于有条件的公司和学校,参照书中介绍的搭建过程,同样可以将实践平台搭建在多台实体计算机上,以便更加接近于大数据和机器学习真实的运行环境。

本书非常适合于学习大数据基础知识的初学者阅读,更适合正在学习大数据理论和技术的人员作为上机实践用的教材。

本书为博硕文化股份有限公司授权出版发行的中文简体字版本

北京市版权局著作权合同登记号:图字 01-2016-7640

本书封面贴有清华大学出版社防伪标签,无标签者不得销售

版权所有,侵权必究。侵权举报电话:010-62782989 13701121933

图书在版编目(CIP)数据

Hadoop + Spark 大数据巨量分析与机器学习整合开发实战 / 林大贵著. — 北京:清华大学出版社,2017
ISBN 978-7-302-45375-8

I. ①H… II. ①林… III. ①数据处理软件 IV. ①TP274

中国版本图书馆 CIP 数据核字(2016)第 260890 号

责任编辑:夏毓彦

封面设计:王 翔

责任校对:闫秀华

责任印制:沈 露

出版发行:清华大学出版社

网 址: <http://www.tup.com.cn>, <http://www.wqbook.com>

地 址:北京清华大学学研大厦 A 座

邮 编:100084

社总机:010-62770175

邮 购:010-62786544

投稿与读者服务:010-62776969, c-service@tup.tsinghua.edu.cn

质 量 反 馈:010-62772015, zhiliang@tup.tsinghua.edu.cn

印 装 者:北京鑫丰华彩印有限公司

经 销:全国新华书店

开 本:190mm×260mm

印 张:27.75

字 数:730 千字

版 次:2017 年 1 月第 1 版

印 次:2017 年 1 月第 1 次印刷

印 数:1~3000

定 价:79.00 元

产品编号:069535-01

序

大数据的影响力正深入到各个领域和行业。特别在商业、经济以及其他领域，将大量数据进行分析后，便可得到许多数据的关联性。这些关联性可用于预测商业趋势、营销研究、金融财务、疾病研究、打击犯罪等。大数据对每一个企业的决策方式将发生变革——决策方式将基于数据和分析的结果，而不是依靠经验和直觉。

信息科技（Information Technology, IT）浪潮的第一波是大型计算机，第二波是个人计算机（PC 机），第三波是网络，第四波是社交媒体，第五波则是“大数据”。每一波的信息科技浪潮都会带来工作与生活方式的改变，创造大量商机、新的产业、大量的工作机会。例如，在网络时代，创造了淘宝、百度、Google（谷歌）、Amazon（亚马逊）等大公司，以及无数.com 公司。

每一波浪潮开始时，相关人才的需求激增，从而造成相关人才的紧缺。因此对个人而言，如果能在浪潮兴起时就投入，往往成果很丰硕，并且有机会占有重要职位。例如，网络刚兴起时，每个公司都需要建立网站，但是这方面的人才当时相对不够，能掌握编写网页相关应用程序设计语言的工程师就能够获得高薪。之后，投入的人越来越多，这方面的工程师就没有当初那么吃香了。

之前的科技浪潮，也许你没有机会躬逢其盛，或是没有机会在浪潮初期进入。而目前大数据的浪潮方兴未艾，正是进入的好时机。根据 IBM 公司调查预估，大数据在 2014 年的市场规模为 71 亿美元，2015 年则达到了 180 亿美元，并将以每年增长 20% 的速度持续成长。机会是给有准备的人的，学会了大数据分析的相关技能，让你有机会获得更好的薪资与职业发展前景。根据美国调查机构 Robert Half Technology 2016 年趋势报告，在美国，大数据工程师的薪水年增长 8.9%，年薪大约 13 万至 18 万美金（约合人民币 85 万元~120 万元）。因为人才短缺，企业不惜重金挖角。（搜索 Robert Half Technology 2016 就可以下载此调查报告。）

本书的主题是 Hadoop+Spark 大数据分析 with 机器学习。众所周知，Hadoop 是运用最多的大数据平台，然而 Spark 异军突起，与 Hadoop 兼容而且运行速度更快，各大公司也开始加入 Spark 的开发。例如，IBM 公司加入 Apache Spark 社区，打算培育百万名数据科学家。谷歌（Google）公司与微软公司也分别应用了 Spark 的功能来构建服务、发展大数据分析云与机器学习平台。这些大公司的加入，也意味着未来更多公司会采用 Hadoop+Spark

进行大数据的数据分析。

然而，目前市面上虽然很多大数据的书，但是多半偏向理论或应用层面的介绍，网络上的信息虽然很多，但是也很杂乱。本书希望能够用浅显易懂的原理介绍和说明，再加上上机实践操作、范例程序，来降低大数据技术的学习门槛，带领读者进入大数据与机器学习的领域。当然整个大数据的生态系非常庞大，需要学习的东西太多。希望读者通过本书的学习，有了基本的概念后，能比较容易踏入这个领域，以便继续深入与研究其他大数据的相关技术。

林大贵

推荐序

如同本书作者所说的，信息技术已经来到了第五波浪潮——“大数据”，在因特网、社交媒体、电子商务等交叉发展和呼应下，“网络”这个巨人已经拥有了难以计数的海量数据，有传统结构化的数据、半结构化的数据，但更多的是非结构化的数据。这些貌似杂乱无章、毫无意义的海量数据，却是一座等待发掘的巨大“金矿”。

这些海量数据中蕴含着极为丰富的人类知识库，它是一笔巨大的信息资产。这些原本很难收集整理的大数据，随着云计算时代的来临，对它们进行及时甚至是实时分析和处理并加以有效利用，就不再是“海市蜃楼”了。

与大数据相关的内容中，不外乎三个方面：大数据理论，大数据分析和处理的技术，大数据的实践应用。目前与大数据有关的出版物中，偏重于理论教学和技术介绍一类的比较多，而偏重于上机实践和自我学习的书却比较少见。因此，本书非常适合大数据学习的初学者和正在学习大数据理论和技术的人员作为上机实践用的教材。

本书从浅显易懂的“大数据和机器学习”原理介绍和说明开始，介绍大数据和机器学习——分类、分析、训练、建模、预测——机器学习（推荐引擎）、机器学习（二元分类）、机器学习（多元分类）、机器学习（回归分析）和数据可视化应用。

在本书中，不是对这些原理进行纯理论的阐述，而是提供了丰富的上机实践操作和范例程序，这样极大地降低了读者学习大数据技术的门槛，对于需要直接上机实践的学习者而言，本书更像是一本大数据学习的实践上机手册。书中首先展示了如何在单台 Windows 系统上通过 Virtual Box 虚拟机安装多台 Linux 虚拟机，而后建立 Hadoop 集群，再建立 Spark 开发环境。搭建这个上机实践的平台并不限制于单台实体计算机，主要是考虑个人读者上机实践的实际条件和环境。对于有条件的公司和学校，参照这个搭建过程，同样可以将实践平台搭建在多台实体计算机上。

在搭建好大数据上机实践的软硬件环境之后，就可以在各个章节的学习中结合本书提供的范例程序逐一设置、修改、调试和运行，从中学到大数据实践应用中核心技术的真谛——对大数据进行高效的“加工”，萃取大数据中蕴含的“智能和知识”，实现数据的“增值”，并最终将其应用于实际工作或者商业中。

资深架构师 赵军

2016年7月

本书章节与范例程序介绍

本书特色

本书的特色是提供了大量上机实践操作与范例程序。

➤ 上机实践操作

一般人可能会认为大数据需要很多台机器的环境才能学习,但是通过本书介绍使用 Virtual Box 虚拟机的方法,就能在自家的计算机上演练建立 Hadoop 集群,并且建立 Spark 开发环境。同时,上机实践操作介绍了 Hadoop MapReduce 与 HDFS 的基本概念,以及 Spark RDD 与 MapReduce 的基本概念。

➤ 范例程序

以实际范例程序来学习程序设计是最有效率的学习方式。因此本书使用实际的数据集,配合范例程序代码来介绍各种机器学习的算法,并示范如何获取数据、训练数据、建立模型、预测结果,由浅入深地介绍 Spark 机器学习。

本书章节内容及上机实践操作与范例程序介绍

➤ 基本概念

章节名称	说明
第 1 章 大数据与机器学习	介绍大数据、Hadoop、HDFS、MapReduce、Spark、机器学习

➤ Hadoop 的安装

章节名称	说明
第 2 章 Virtual Box 虚拟机软件的安装	上机实践操作 安装 Virtual Box 虚拟机,让你可以在 Windows 系统上安装多台 Linux 虚拟机
第 3 章 Ubuntu Linux 操作系统的安装	上机实践操作 安装 Ubuntu Linux 操作系统

(续表)

章节名称	说明
第 4 章 Hadoop Single Node Cluster 的安装	上机实践操作 安装单台机器的 Hadoop Single Node Cluster
第 5 章 Hadoop Multi Node Cluster 的安装	上机实践操作 安装多台机器的 Hadoop Multi Node Cluster

➤ **Hadoop 的基本功能**

章节名称	说明
第 6 章 Hadoop HDFS 命令	上机实践操作 示范如何使用 HDFS 命令
第 7 章 Hadoop MapReduce	介绍 Hadoop MapReduce 的原理 WordCount.java 范例程序 示范使用 Hadoop MapReduce 计算文章内的每一个单词出现的次数

➤ **Spark 的基本功能**

章节名称	说明
第 8 章 Spark 的安装与介绍	上机实践操作 Spark 安装与 spark-shell 交互界面在不同环境中的运行示范
第 9 章 Spark RDD	上机实践操作 介绍 Spark 最基本的功能 RDD (Resilient Distributed Dataset, 弹性分布式数据集) 的基本运算
第 10 章 Spark 的集成开发环境	上机实践操作 安装集成开发环境 (IDE) WordCount.scala 范例程序 示范使用 Spark MapReduce 计算文章内的每一个单词出现的次数

➤ **机器学习 (推荐引擎)**

章节名称	说明
第 11 章创建推荐引擎	介绍如何使用 Spark MLlib 以 MovieLens 数据集建立电影的推荐引擎 (Recommendation Engine) Recommend.scala 范例程序 示范如何获取数据、训练模型、推荐用户或电影, 建立电影的推荐系统 AlsEvaluation.scala 范例程序 示范如何调试推荐引擎参数, 找出最佳的参数组合

➤ 机器学习（二元分类）

章节名称	说明
第 12 章 StumbleUpon 数据集	StumbleUpon 数据集属于二元分类问题，可以根据网页的特征预测哪些网页是暂时性的或是可以长久存在的
第 13 章决策树二元分类	RunDecisionTreeBinary.scala 范例程序 示范如何使用决策树二元分类分析 StumbleUpon 数据集，预测哪些网页是暂时性的或可以长久存在的，并且找出最佳的参数组合，提高预测准确度
第 14 章逻辑回归二元分类	RunLogisticRegressionWithSGDBinary.scala 范例程序 示范如何使用决策树二元分类分析 StumbleUpon 数据集，预测哪些网页是暂时性的或是可以长久存在的，并且找出最佳的参数组合，提高预测准确度
第 15 章支持向量机 SVM 二元分类	RunSVMWithSGDBinary.scala 范例程序 示范如何使用支持向量机 SVM 二元分类分析 StumbleUpon 数据集，预测哪些网页是暂时性的或是可以长久存在的，并且找出最佳的参数组合，提高预测准确度
第 16 章朴素贝叶斯二元分类	RunNaiveBayesBinary.scala 范例程序 示范如何使用朴素贝叶斯（Naïve-Bayes）二元分类分析 StumbleUpon 数据集，预测哪些网页是暂时性的或是可以长久存在的，并且找出最佳的参数组合，提高预测准确度

➤ 机器学习（多元分类）

章节名称	说明
第 17 章决策树多元分类	RunDecisionTreeMulti.scala 范例程序 示范如何使用决策树多元分类分析 Covtype 数据集（森林覆盖植被），根据不同的土地条件可以预测该地的植被，并且找出最佳的参数组合，提高预测准确度

➤ 机器学习（回归分析）

章节名称	说明
第 18 章决策树回归分析	RunDecisionTreeRegression.scala 范例程序 示范介绍决策树回归分析，分析 Bike Sharing 数据集。根据天气和假日条件，可以预测每一小时租借的数量，并且找出最佳的参数组合，提高预测准确度

➤ 数据可视化

章节名称	说明
第 19 章使用 Apache Zeppelin 数据可视化	上机实践操作 安装 Zeppelin 并使用 ml-100k 数据集，示范使用 Spark SQL 进行数据分析与数据可视化

本书上机实践操作命令的整理

本书从第 2 章到第 10 章，我们使用了很多 Linux、spark-shell、SparkSQL 等命令。不过很多命令都很长，只要有一个字母打错就无法运行，这样会增加学习的挫折感。因此我们在博客文章中整理了各个章节使用的命令。可参考网页：<http://www.weibo.com/hadoopsparkbook>。

安装或练习命令时，你可以复制博客文章中的命令，然后粘贴到“终端”程序中，这样既可以节省打字的时间，也不用担心打错字母（如果无法在 VirtualBox 虚拟机的 Ubuntu “终端”程序中执行复制/粘贴操作，可参考第 3.9 节的说明，设置好 VirtualBox 的共享剪贴板）。

本书范例程序的下载与安装

以实际范例程序来学习程序设计是最有效率学习的方式。因此本书使用实际的数据集，配合范例程序来介绍各种算法，并示范如何获取数据、训练数据、建立模型、预测结果，由浅入深地介绍 Hadoop 与 Spark 机器学习。

你可以到下面的网址下载本书的范例程序：

<http://pan.baidu.com/s/1qYMtjNQ>（注意数字及字母大小写，如果下载有问题，请电子邮件联系 booksaga@126.com，邮件主题为“Hadoop + Spark 大数据巨量分析程序”）

但是，在这些范例程序安装和使用之前，必须建立整个开发环境。建议按照本书的顺序阅读第 2 到 10 章，并且真正建立整个开发环境。本书范例程序的安装说明将在第 10.16 节中介绍。

读者服务与社区交流

在网络时代，我们希望购买本书的读者不只是获得本书的内容，还能通过网络社区获得更多的信息。

➤ 本书的博客

网址：<http://blog.sina.com.cn/hadoopsparkbook>。

我们还建立了本书的博客，将一些需要排列整齐，系统化的信息放在博客文章中。如果有新的博客文章，内容包括：

- (1) 本书上机实践操作命令的整理。
- (2) 本书内容或程序代码的勘误。
- (3) 分享最新的 Hadoop 或 Spark 信息。

➤ 本书的微博

网址：<http://www.weibo.com/hadoopsparkbook>。

我们建立了本书的 Facebook 粉丝团，欢迎读者们加入。粉丝团会不定期贴文，分享最新的 Hadoop 或 Spark 信息，你也可以提问并参与交流。

目 录

第 1 章 大数据与机器学习	1
1.1 大数据定义	2
1.2 Hadoop 简介	2
1.3 Hadoop HDFS 分布式文件系统	3
1.4 Hadoop MapReduce 的介绍	5
1.5 Spark 的介绍	6
1.6 机器学习的介绍	8
第 2 章 VirtualBox 虚拟机软件的安装	11
2.1 VirtualBox 的下载和安装	12
2.2 设置 VirtualBox 语言版本	16
2.3 设置 VirtualBox 存储文件夹	17
2.4 在 VirtualBox 创建虚拟机	18
第 3 章 Ubuntu Linux 操作系统的安装	23
3.1 下载安装 Ubuntu 的光盘文件	24
3.2 在 Virtual 设置 Ubuntu 虚拟光盘文件	26
3.3 开始安装 Ubuntu	28
3.4 启动 Ubuntu	33
3.5 安装增强功能	34
3.6 设置默认输入法	38
3.7 设置“终端”程序	40
3.8 设置“终端”程序为白底黑字	42
3.9 设置共享剪贴板	43

第 4 章 Hadoop Single Node Cluster 的安装	46
4.1 安装 JDK	47
4.2 设置 SSH 无密码登录	50
4.3 下载安装 Hadoop	53
4.4 设置 Hadoop 环境变量	56
4.5 修改 Hadoop 配置设置文件	58
4.6 创建并格式化 HDFS 目录	62
4.7 启动 Hadoop	63
4.8 打开 Hadoop ResourceManager Web 界面	66
4.9 NameNode HDFS Web 界面	67
第 5 章 Hadoop Multi Node Cluster 的安装	69
5.1 把 Single Node Cluster 复制到 data1	71
5.2 设置 VirtualBox 网卡	73
5.3 设置 data1 服务器	76
5.4 复制 data1 服务器到 data2、data3、master	84
5.5 设置 data2、data3 服务器	87
5.6 设置 master 服务器	91
5.7 master 连接到 data1、data2、data3 创建 HDFS 目录	94
5.8 创建并格式化 NameNode HDFS 目录	98
5.9 启动 Hadoop Multi Node Cluster	99
5.10 打开 Hadoop ResourceManager Web 界面	102
5.11 打开 NameNode Web 界面	103
第 6 章 Hadoop HDFS 命令	104
6.1 启动 Hadoop Multi-Node Cluster	105
6.2 创建与查看 HDFS 目录	107
6.3 从本地计算机复制文件到 HDFS	109
6.4 将 HDFS 上的文件复制到本地计算机	114
6.5 复制与删除 HDFS 文件	116
6.6 在 Hadoop HDFS Web 用户界面浏览 HDFS	118
第 7 章 Hadoop MapReduce	122
7.1 介绍 wordCount.Java	123

7.2	编辑 wordCount.Java	124
7.3	编译 wordCount.Java	127
7.4	创建测试文本文件	129
7.5	运行 wordCount.Java	130
7.6	查看运行结果	131
7.7	Hadoop MapReduce 的缺点	132
第 8 章	Spark 的安装与介绍	133
8.1	Spark 的 Cluster 模式架构图	134
8.2	Scala 的介绍与安装	135
8.3	安装 Spark	138
8.4	启动 spark-shell 交互界面	141
8.5	设置 spark-shell 显示信息	142
8.6	启动 Hadoop	144
8.7	本地运行 spark-shell 程序	145
8.8	在 Hadoop YARN 运行 spark-shell	147
8.9	构建 Spark Standalone Cluster 执行环境	149
8.10	在 Spark Standalone 运行 spark-shell	155
第 9 章	Spark RDD	159
9.1	RDD 的特性	160
9.2	基本 RDD “转换” 运算	161
9.3	多个 RDD “转换” 运算	167
9.4	基本 “动作” 运算	169
9.5	RDD Key-Value 基本 “转换” 运算	171
9.6	多个 RDD Key-Value “转换” 运算	175
9.7	Key-Value “动作” 运算	178
9.8	Broadcast 广播变量	181
9.9	accumulator 累加器	184
9.10	RDD Persistence 持久化	186
9.11	使用 Spark 创建 WordCount	188
9.12	Spark WordCount 详细解说	191

第 10 章 Spark 的集成开发环境	195
10.1 下载与安装 eclipse Scala IDE.....	197
10.2 下载项目所需要的 Library	201
10.3 启动 eclipse	205
10.4 创建新的 Spark 项目	206
10.5 设置项目链接库	210
10.6 新建 scala 程序	211
10.7 创建 WordCount 测试文本文件	213
10.8 创建 WordCount.scala	213
10.9 编译 WordCount.scala 程序	215
10.10 运行 WordCount.scala 程序	217
10.11 导出 jar 文件.....	220
10.12 spark-submit 的详细介绍	223
10.13 在本地 local 模式运行 WordCount 程序	224
10.14 在 Hadoop yarn-client 运行 WordCount 程序	226
10.15 在 Spark Standalone Cluster 上运行 WordCount 程序	230
10.16 本书范例程序的安装说明	231
第 11 章 创建推荐引擎	236
11.1 推荐算法介绍	237
11.2 “推荐引擎”大数据分析使用场景	237
11.3 ALS 推荐算法的介绍	238
11.4 ml-100k 推荐数据的下载与介绍	240
11.5 使用 spark-shell 导入 ml-100k 数据	242
11.6 查看导入的数据	244
11.7 使用 ALS.train 进行训练	247
11.8 使用模型进行推荐	250
11.9 显示推荐的电影名称	252
11.10 创建 Recommend 项目	255
11.11 Recommend.scala 程序代码	257
11.12 创建 PrepareData()数据准备	259
11.13 recommend()推荐程序代码	261
11.14 运行 Recommend.scala	263
11.15 创建 AlsEvaluation.scala 调校推荐引擎参数	266

11.16	创建 PrepareData()数据准备	269
11.17	进行训练评估	270
11.18	运行 AlsEvaluation	279
11.19	修改 Recommend.scala 为最佳参数组合	281
第 12 章	StumbleUpon 数据集	282
12.1	StumbleUpon 数据集简介	283
12.2	下载 StumbleUpon 数据	285
12.3	用 LibreOffice Calc 电子表格查看 train.tsv	288
12.4	二元分类算法	291
第 13 章	决策树二元分类	292
13.1	决策树的介绍	293
13.2	创建 Classification 项目	294
13.3	开始输入 RunDecisionTreeBinary.scala 程序	296
13.4	数据准备阶段	298
13.5	训练评估阶段	303
13.6	预测阶段	308
13.7	运行 RunDecisionTreeBinary.scala	311
13.6	修改 RunDecisionTreeBinary 调校训练参数	313
13.7	运行 RunDecisionTreeBinary 进行参数调校	320
13.8	运行 RunDecisionTreeBinary 不进行参数调校	323
第 14 章	逻辑回归二元分类	326
14.1	逻辑回归分析介绍	327
14.2	RunLogisticRegression WithSGDBinary.scala 程序说明	328
14.3	运行 RunLogisticRegression WithSGDBinary.scala 进行参数调校	331
14.4	运行 RunLogisticRegression WithSGDBinary.scala 不进行参数调校	335
第 15 章	支持向量机 SVM 二元分类	337
15.1	支持向量机 SVM 算法的基本概念	338
15.2	RunSVMWithSGDBinary.scala 程序说明	338
15.3	运行 SVMWithSGD.scala 进行参数调校	341
15.4	运行 SVMWithSGD.scala 不进行参数调校	344

第 16 章 朴素贝叶斯二元分类	346
16.1 朴素贝叶斯分析原理的介绍	347
16.2 RunNaiveBayesBinary.scala 程序说明	348
16.3 运行 NaiveBayes.scala 进行参数调校	351
16.4 运行 NaiveBayes.scala 不进行参数调校	353
第 17 章 决策树多元分类	355
17.1 “森林覆盖植被”大数据问题分析场景	356
17.2 UCI Coverttype 数据集介绍	357
17.3 下载与查看数据	359
17.4 创建 RunDecisionTreeMulti.scala	361
17.5 修改 RunDecisionTreeMulti.scala 程序	362
17.6 运行 RunDecisionTreeMulti.scala 进行参数调校	367
17.7 运行 RunDecisionTreeMulti.scala 不进行参数调校	371
第 18 章 决策树回归分析	373
18.1 Bike Sharing 大数据问题分析	374
18.2 Bike Sharing 数据集	375
18.3 下载与查看数据	375
18.4 创建 RunDecisionTreeRegression.scala	378
18.5 修改 RunDecisionTreeRegression.scala	380
18.6 运行 RunDecisionTreeRegression.scala 进行参数调校	389
18.7 运行 RunDecisionTreeRegression.scala 不进行参数调校	392
第 19 章 使用 Apache Zeppelin 数据可视化	394
19.1 Apache Zeppelin 简介	395
19.2 安装 Apache Zeppelin	395
19.3 启动 Apache Zeppelin	399
19.4 创建新的 Notebook	402
19.5 使用 Zeppelin 运行 Shell 命令	403
19.6 创建临时表 UserTable	406
19.7 使用 Zeppelin 运行年龄统计 Spark SQL	407
19.8 使用 Zeppelin 运行性别统计 Spark SQL	409
19.9 按照职业统计	410

19.10	Spark SQL 加入文本框输入参数	412
19.11	加入选项参数	414
19.12	同时显示多个统计字段	416
19.13	设置工具栏	419
19.14	设置段落标题	420
19.15	设置 Paragraph 段落的宽度	422
19.16	设置显示模式	423

第 1 章

大数据与机器学习

- 1.1 大数据定义
- 1.2 Hadoop 简介
- 1.3 Hadoop HDFS 分布式文件系统
- 1.4 Hadoop MapReduce 的介绍
- 1.5 Spark 的介绍
- 1.6 机器学习的介绍

1.1 大数据定义

大数据 (Big data) 又称为巨量资料、巨量数据或海量数据。一般来说大数据的特性可归类为 3V:

➤ Volume (大量数据)

- **因特网、企业 IT、物联网、社区、短信、电话、网络搜索、在线交易等**，随时都在快速累积庞大的数据。
- **数据量很容易达到 TB (Terabyte, 1024 GB)**，甚至 PB (Petabyte, 1024TB) 或 EB (Exabyte, 1024 PB) 的等级。

➤ Variety (多样性)

大数据的数据类型非常多样化，可分为非结构化信息和结构化信息。

- **非结构化信息**：文字、图片、图像、视频、音乐、地理位置信息、个人化信息——如社区、交友数据等。
- **结构化信息**：数据库、数据仓库等。

➤ Velocity (时效性)

- **数据的传输流动**：随着带宽越来越大、设备越来越多，每秒产生的数据流越来越大。
- **组织必须能实时处理大量的信息**：因为时间太久就失去了数据的价值，数据必须能在最短时间内得出分析结果。

大数据的影响已经深入到各个领域和行业，在商业、经济及其他领域中，将大量数据进行分析后，就可得出许多数据的关联性。可用于预测商业趋势、营销研究、金融财务、疾病研究、打击犯罪等。决策行为将基于数据和分析的结果，而不是依靠经验和直觉。

1.2 Hadoop 简介

Hadoop 是存储与处理大量数据的平台，是 Apache 软件基金会的开放源码、免费且广泛使用的软件。Hadoop 名称来自于原作者孩子的黄色小象玩具。这个名字容易拼字与发音，适合作为软件开发代码，黄色小象因此成为 Hadoop 的标志。

➤ Hadoop 的发展历史

2002 年 Doug Cutting 与 Mike Cafarella 开始进行 Nutch 项目。

2003 年 Google 发表 GFS (Google File System) 与 MapReduce 论文。

2004 年 Doug Cutting 开始将 DFS 与 MapReduce 加入 Nutch 项目。

2006 年 Doug Cutting 加入 Yahoo 团队，并将 Nutch 改名为 Hadoop。

2008 年 Yahoo 使用 Hadoop 包含了 910 个集群，对 1TB 的数据排序花了 297 秒。

➤ Hadoop 特性

特性	说明
可扩展性 (Scalable)	由于 Hadoop 采用分布式计算与存储，当我们扩充容量或运算时，不需要更换整个系统，只需要增加新的数据节点服务器即可
经济性 (Economical)	由于 Hadoop 采用分布式计算与存储，不必使用昂贵高端的服务器，只需一般等级的服务器就可架构出高性能、高容量的集群
弹性 (Flexible)	传统的关系数据库存储数据时必须有数据表结构 schema（各个数据对象的集合），然而 Hadoop 存储的数据是非结构化（schema-less）的，也就是说可以存储各种形式、不同数据源的数据
可靠性 (Reliable)	Hadoop 采用分布式架构，因此即使某一台服务器硬件坏掉，甚至整个机架坏掉，HDFS 仍可正常运行，因为数据还会有另外 2 个副本

1.3 Hadoop HDFS 分布式文件系统

HDFS 采用分布式文件系统（Hadoop Distributed File System），可以由单台服务器扩充到数千台服务器，如图 1-1 所示。

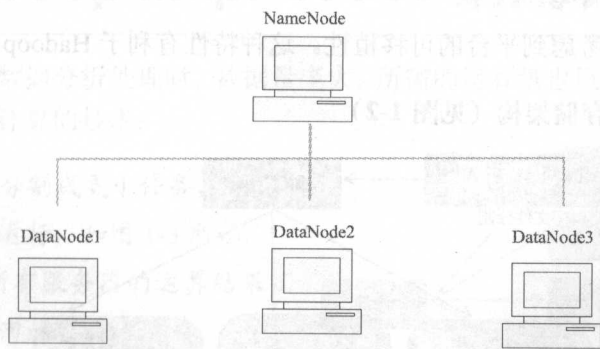


图 1-1 HDFS 采用分布式文件系统的可扩充架构

- NameNode 服务器负责管理与维护 HDFS 目录系统并控制文件的读写操作。
- 多个 DataNode 服务器负责存储数据，在图 1-1 中我们只列出 3 个 DataNode，实际上大型的集群可以有成千上万个节点。

➤ HDFS 设计的前提与目标

- 硬件故障是常态而不是异常（Hardware Failure）

HDFS 是设计运行在低成本的普通服务器上的。硬件故障是常态，而不是异常，所以 HDFS 被设计成具有高度容错能力，能够实时检测错误并且自动恢复，这是 HDFS 最核心的设计目标。

● Streaming 流式数据存取 (Streaming Data Access)

运行在 HDFS 上的应用程序会通过 Streaming 存取数据集。HDFS 的主要设计是批处理，而不是实时互动处理，优点是可以提高存取大量数据的能力，但是牺牲了响应时间。

● 大数据集 (Large Data Sets)

为了存储大数据集，HDFS 提供了 cluster 集群架构，用于存储大数据文件，集群可扩充至数百个节点。

● 简单一致性模型 (Simple Coherency Model)

HDFS 的存取模式是一次写入多次读取 (write-once-read-many)。一个文件被创建后就不会再修改。这样设计的优点是：可以提高存储大量数据的能力，并简化数据一致性的问题。

● 移动“计算”比移动“数据”成本更低 (Moving Computation is Cheaper than Moving Data)

当我们的 cluster 集群存储了大量的数据时，要搬移数据必须耗费大量的时间成本。因此如果我们有“计算”数据的需求时，就会将“计算功能”在接近数据的服务器中运行，而不是搬移数据。

● 跨硬件与软件平台

HDFS 在设计时就考虑到平台的可移植性。这种特性有利于 Hadoop 的推广。

➤ HDFS 文件存储架构 (见图 1-2)

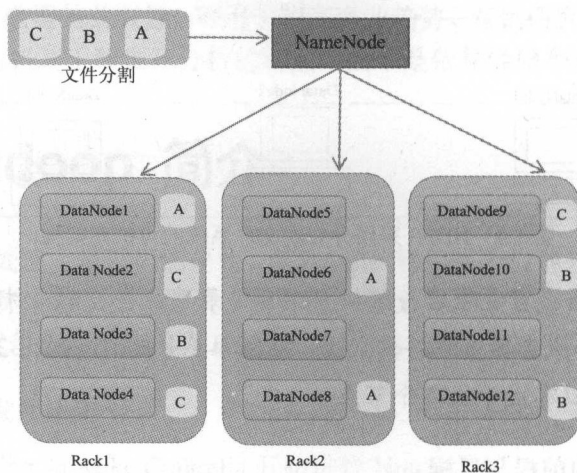


图 1-2 HDFS 文件存储架构

➤ 文件分割

- 当用户以 HDFS 命令要求存储文件时，系统会将文件切割为多个区块（Block），每个区块是 64MB。在图 1-2 中，文件被分割为 A、B、C 共 3 个区块。

➤ 区块副本策略

- 一个文件区块默认会复制成 3 份，你可以在 Hadoop 配置中设置文件区块要创建几个副本。
- 文件区块损坏时，NameNode 会自动寻找位于其他 DataNode 上的副本来恢复数据，维持 3 份的副本策略。

➤ 机架感知

- 在图 1-2 中，共有 Rack1、Rack2、Rack3 共 3 个机架，每个机架都有 4 台 DataNode 服务器。
- HDFS 具备机架感知功能，如图 1-2 所示。以 Block C 为例：第一份副本放在 Rack1 机架的节点，第二份放在同机架 Rack1 的不同节点，最后一份放在不同机架 Rack3 的不同节点。
- 这样设计的好处是防止数据遗失，有任何机架出现故障仍可以保证恢复数据，提高网络性能。

1.4 Hadoop MapReduce 的介绍

利用大数据进行数据分析处理时，数据量庞大，所需的运算量也巨大。Hadoop MapReduce 的做法是采用分布式计算的技术：

- Map 将任务分割成更小任务，由每台服务器分别运行，如图 1-3 所示。
- Reduce 将所有服务器的运算结果汇总整理，返回最后的结果。

通过 MapReduce 方式，可以在上千台机器上并行处理巨量的数据，大大减少数据处理的时间。

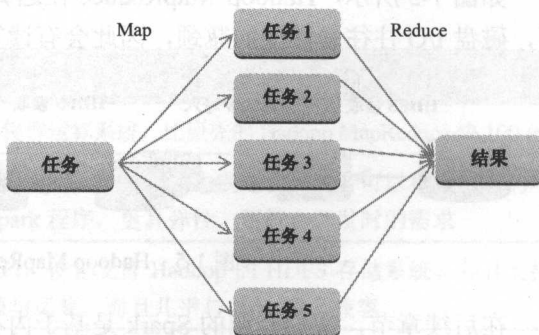


图 1-3 Hadoop MapReduce 运行的示意图

➤ MapReduce 版本 2.0 YARN

Hadoop 新的 MapReduce 架构称为 YARN（Yet Another Resource Negotiator，另一种资源协调者），是效率更高的资源管理核心。可以到下列网址查看 YARN 架构图（见图 1-4）：

<http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html>

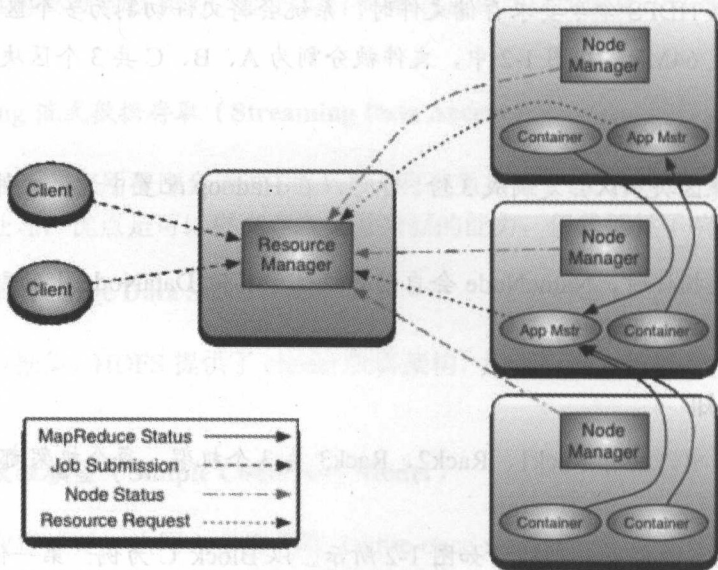


图 1-4 YARN 资源管理系统的架构图

从图 1-4 中可以看到：

- 在 Client 客户端，用户会向 Resource Manager 请求执行运算（或执行任务）。
- 在 NameNode 会有 Resource Manager 统筹管理运算的请求。
- 在其他的 DataNode 会有 Node Manager 负责运行，以及监督每一个任务（task），并且向 Resource Manager 汇报状态。

➤ Hadoop MapReduce 的计算框架

如图 1-5 所示，Hadoop MapReduce 在运算时，需要将中间产生的数据存储在硬盘中。然而，磁盘 I/O 往往是性能的瓶颈，因此会有读写数据延迟的问题。

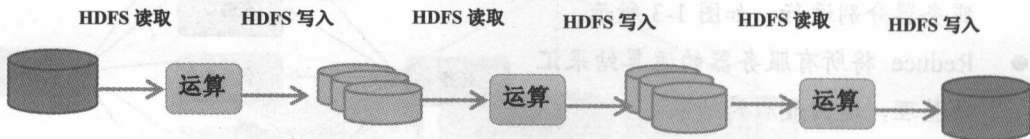


图 1-5 Hadoop MapReduce 运算过程的示意图

在后续章节，我们介绍的 Spark 是基于内存的计算框架，可以大幅提升性能。

1.5 Spark 的介绍

Apache Spark 是开放源码的集群运算框架，由加州大学伯克利分校的 AMPLab 开发。Spark

是一个弹性的运算框架，适合进行 Spark Streaming 数据流处理、Spark SQL 互动分析、MLlib 机器学习等应用，因此 Spark 可成为一个用途广泛的大数据运算平台。Spark 允许用户将数据加载到 cluster 集群的内存中存储，并多次重复运算，非常适合用于机器学习的算法。

➤ Spark in-memory 的计算框架

如图 1-6 所示，Spark 是基于内存的计算框架。Spark 在运算时，将中间产生的数据暂存在内存中，因此可以加快运行速度。尤其是需要反复操作的次数越多，所需读取的数据量越大，就越能看出 Spark 的性能。Spark 在内存中运行程序，命令运行速度（或命令周期）能比 Hadoop MapReduce 的命令运行速度快上 100 倍，即便是运行于硬盘上时 Spark 的速度也能快上 10 倍。

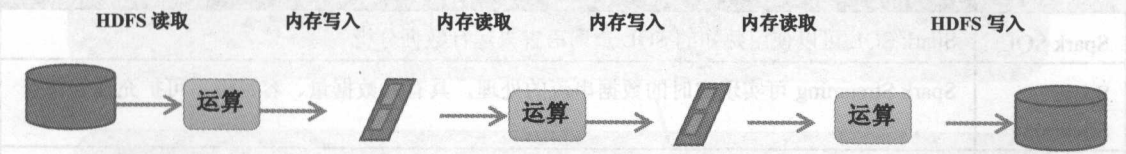


图 1-6 Spark in-memory 的计算框架

➤ Spark 发展历史

年代	说明
2009	由 Matei Zaharia 在加州大学伯克利分校的 AMPLab 开发
2010	通过 BSD 授权条款发布开放源码
2013	该项目被捐赠给 Apache 软件基金会
2014/2	Spark 成为 Apache 的顶级项目
2014/11	Databricks 团队使用 Spark 刷新数据排序的世界纪录
2015/3	Spark 1.3.1 版发布

➤ Spark 特色

特色	说明
命令周期短	Spark 是基于内存计算的开放源码集群运算系统，比原先的 Hadoop MapReduce 快 100 倍
易于开发程序	目前 Spark 支持多种语言：Scala、Python、Java，也就是说开发者可以根据应用的环境来决定使用哪一种语言开发 Spark 程序，更具弹性，更符合开发时的需求
Hadoop 兼容	Spark 提供了 Hadoop Storage API，使它支持 Hadoop 的 HDFS 存储系统。并且支持 Hadoop YARN，可共享存储资源与运算，而且几乎与 Hive 完全兼容
可在各平台运行	<ul style="list-style-type: none">我们可以在本地端的机器上运行 Spark 程序，只需要 import Spark 的链接库即可也可以在自有的群集上运行 Spark 程序，例如 Mesos、Hadoop YARN 等自行建立的群集针对更大规模的计算工作，我们可以选择将 Spark 程序送至 AWS 的 EC2 平台上运行，按照用户使用的计算资源计费

➤ Spark 主要功能（参考图 1-7）

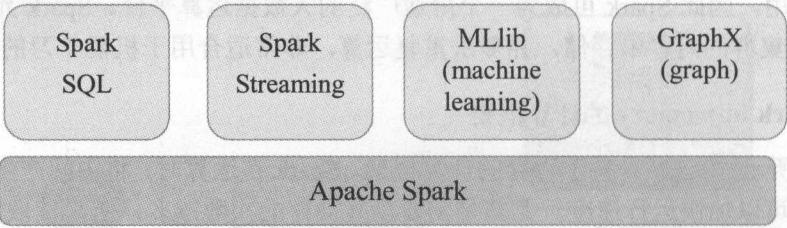


图 1-7 Spark 的主要功能模块

功能	说明
Spark SQL	Spark SQL 可以使用熟知的 SQL 查询语言来运行数据分析
Spark Streaming	Spark Streaming 可实现实时的数据串流的处理，具有大数据量、容错性、可扩充性等特点
GraphX	GraphX 是 Spark 上的分布式图形处理架构，可用图表计算
MLlib	MLlib 是一个可扩充的 Spark 机器学习库，可使用许多常见的机器学习算法，简化大规模机器学习的时间。算法包括分类与回归、支持向量机、回归、线性回归、决策树、朴素贝叶斯、聚类分析、协同过滤等

1.6 机器学习的介绍

由于机器学习技术的进步，应用相当广泛，例如推荐引擎、定向广告、需求预测、垃圾邮件过滤、医学诊断、自然语言处理、搜索引擎、欺诈检测、证券分析、视觉识别、语音识别、手写识别等。

➤ 机器学习架构

机器学习（Machine Learning）是通过算法，使用历史数据进行训练，训练完成后会产生模型。未来当有新的数据提供时，我们可以使用训练产生的模型进行预测。

机器学习训练用的数据是由 **Features**、**Label** 组成的。

- **Feature:** 数据的特征，例如湿度、风向、风速、季节、气压。
- **Label:** 数据的标签，也就是我们希望预测的目标，例如降雨（0.不会下雨、1.会下雨）、天气（1.晴天、2.雨天、3.阴天、4.下雪）、气温。

如图 1-8 所示，机器学习可分为以下两个阶段。

● 训练阶段（Training）

训练数据是过去累积的历史数据，可能是文本文件、数据库或其他来源。经过 Feature Extraction（特征提取），产生 Feature（数据特征）与 Label（预测目标），然后经过机器学习

算法的训练后产生模型。

● 预测阶段 (Predict)

新输入数据 (可能是文本文件、数据库或其他来源), 经过 Feature Extraction (特征提取) 产生 Feature, 使用训练完成的模型进行预测, 最后产生预测结果。

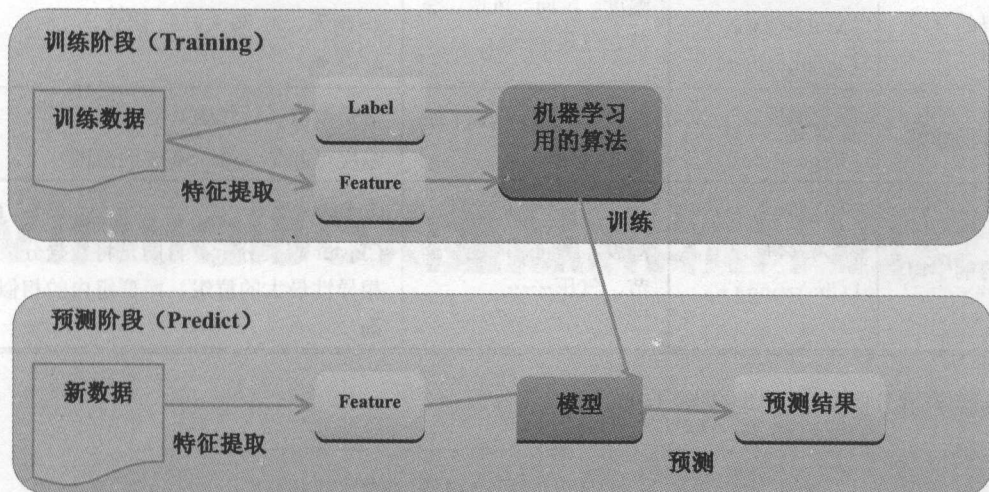


图 1-8 机器学习的两个阶段

➤ 机器学习分类

对于有监督的学习 (Supervised Learning), 从现有数据我们希望预测的答案有下列分类。

● 二元分类

我们已知湿度、风向、风速、季节、气压等数据特征, 希望预测当天是否会下雨 (0. 不会下雨、1. 会下雨)。因为希望预测的目标 Label 只有 2 种选项, 所以就像是非题。

● 多元分类

我们已知湿度、风向、风速、季节、气压等数据特征, 希望预测当天的天气 (1. 晴天、2. 雨天、3. 阴天、4. 下雪)。因为希望预测的目标 Label 有多个选项, 所以就像选择题。

● 回归分析

我们已知湿度、风向、风速、季节、气压等数据特征, 希望预测当天的气温。因为希望预测的目标 Label 是连续值, 所以就像是计算题。

但是对于无监督的学习 (Unsupervised Learning), 从现有数据我们不知道要预测的答案, 所以没有 Label (预测目标)。cluster 聚类分析的目的是将数据分成几个相异性最大的群组, 而群组内的相似性最高。

根据上述内容我们可以整理出下列表格。

分类	算法	Features（特征）	Label（预测目标）
有监督的学习	二元分类 (Binary Classification)	湿度、风向、风速、季节、气压……	只有 0 与 1 选项（是非题） 0. 不会下雨、1. 会下雨
有监督的学习	多元分类 (Multi-Class Classification)	湿度、风向、风速、季节、气压……	有多个选项（选择题） 1. 晴天、2. 雨天、3. 阴天、4. 下雪
有监督的学习	回归分析 (Regression)	湿度、风向、风速、季节、气压……	值是数值（计算题） 温度可能是 -50~50 度的范围
无监督的学习	聚类分析 (Clustering)	湿度、风向、风速、季节、气压……	无 Label Cluster 聚类分析：目的是将数据分成几个相异性最大的群组，而群组内的相似性最高

机器学习分类可以整理成图 1-9。

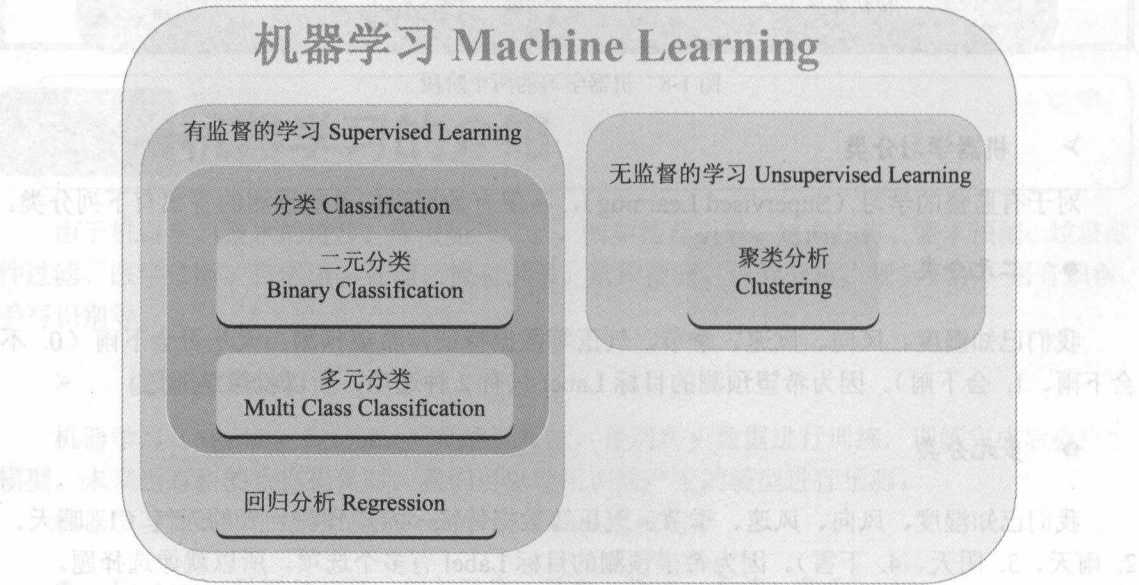


图 1-9 机器学习的分类

VirtualBox虚拟机软件的安装

- 2.1 VirtualBox 的下载和安装
- 2.2 设置 VirtualBox 语言版本
- 2.3 设置 VirtualBox 存储文件夹
- 2.4 在 VirtualBox 创建虚拟机

接下来，我们将介绍 Hadoop 的安装方法。根据 Hadoop 官方说明文件的建议，Hadoop 最主要是在 Linux 操作系统环境下运行。市面上有很多 Open Source 开放源码的操作系统都属于 Linux 操作系统，例如 Fedora、Ubuntu 等。

本书将介绍如何安装 Ubuntu，并且在 Ubuntu 上安装 Hadoop 和 Spark。不过当我们介绍搭建 Hadoop cluster 集群时，则需要多台计算机。但是，大多数的读者使用的是 Windows 操作系统，而且没有很多台计算机。

为了让读者们可以上机实践，我们会介绍如何安装 VirtualBox 虚拟机软件，这样就能够新增多台虚拟机，然后在虚拟机中安装 Ubuntu 与 Hadoop 集群。如果你有多台实体计算机，也可以安装在实体计算机中，安装的方法与虚拟机相同。

VirtualBox 是由 Oracle 公司所开发的，是一款免费并且具有中文版的虚拟机（Virtual Machine）软件，目前流行的是 5.0 版本。VirtualBox 很适合练习操作系统和软件的安装与测试，在虚拟计算机中进行的任何实验都不会影响实体计算机的正常运行。安装 VirtualBox 之后，你可以在计算机新增多台虚拟机，然后在虚拟机中安装不同的操作系统，例如 Windows、Linux 等，使用上与实体计算机一样。

2.1 VirtualBox 的下载和安装

关于 VirtualBox 5.0 的下载、安装以及设置说明如下。

步骤 01 VirtualBox 下载网址

连接到网址，选择“x86/amd64”，下载 VirtualBox Windows 版本，如图 2-1 所示。

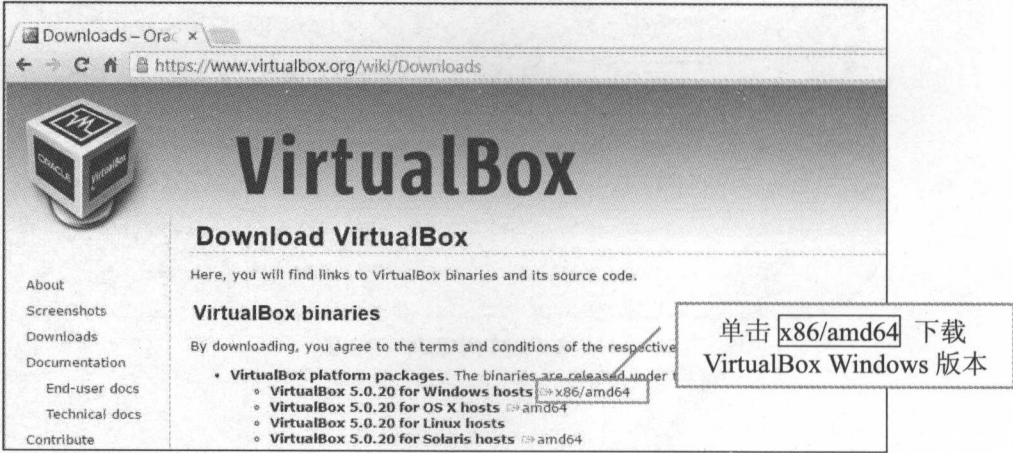


图 2-1 从 VirtualBox 官网下载 VirtualBox 5.0

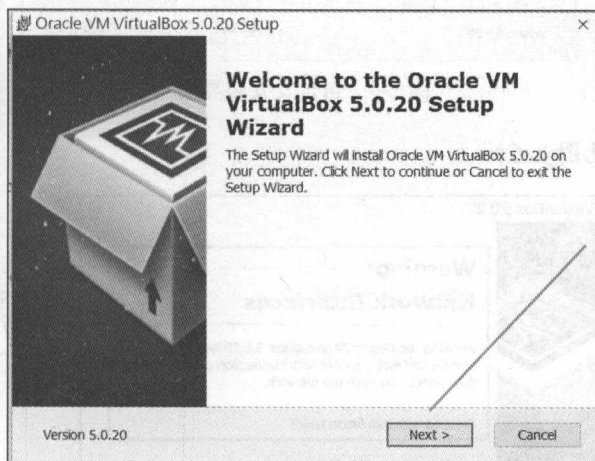
步骤 02 运行 VirtualBox 安装程序（见图 2-2）



下载后在存储的目录中可以看到已下载的安装文件，用鼠标左键双击安装程序的图标，即可开始运行安装程序

图 2-2 开始安装程序

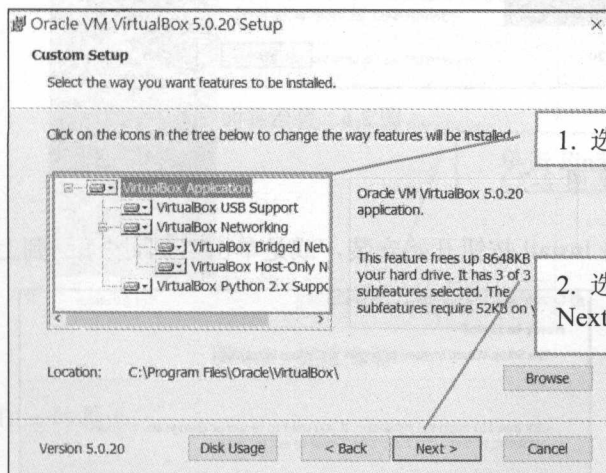
步骤 03 开始安装 VirtualBox (见图 2-3)



单击 Next 按钮开始安装

图 2-3 单击 Next 按钮开始安装

步骤 04 选择 VirtualBox 功能 (见图 2-4)



1. 选择要安装的功能

2. 选择默认安装，单击 Next 按钮即可

图 2-4 选择安装的功能或者选择默认安装

步骤 05 自定义安装界面 (见图 2-5)

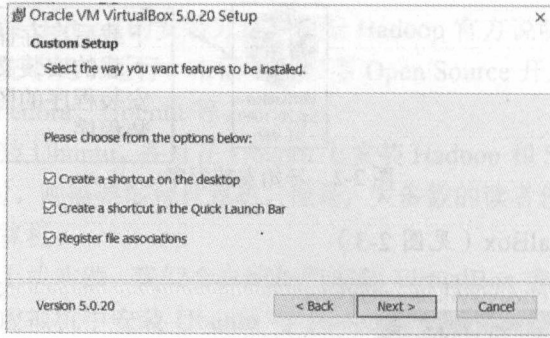


图 2-5 自定义安装界面

步骤 06 警告界面（见图 2-6）

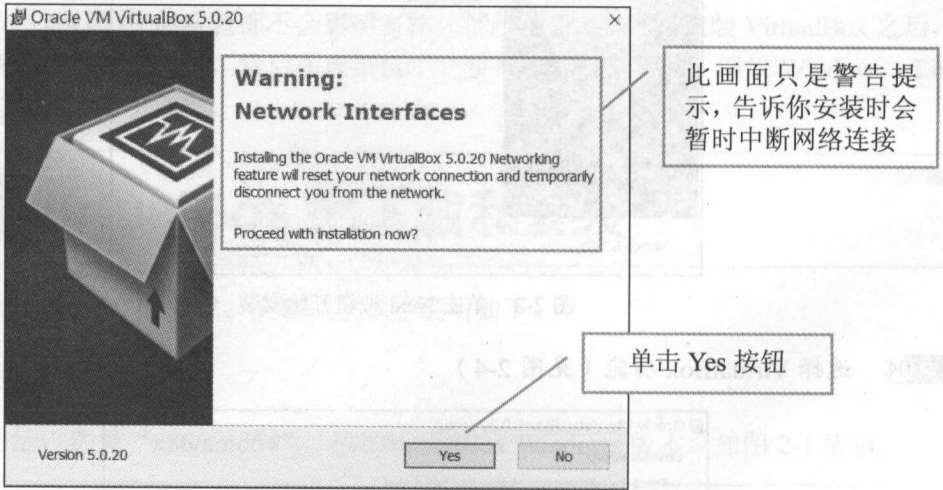


图 2-6 警告界面

步骤 07 完成设置（见图 2-7）

完成设置后，单击 Install 按钮开始安装，或是单击 Back 按钮回到上一个步骤修改设置。

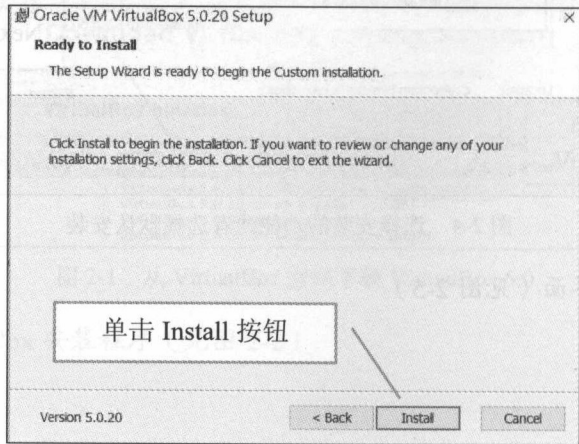


图 2-7 完成设置开始实际的安装

步骤 08 开始安装

开始进行安装的屏幕显示界面如图 2-8 所示。

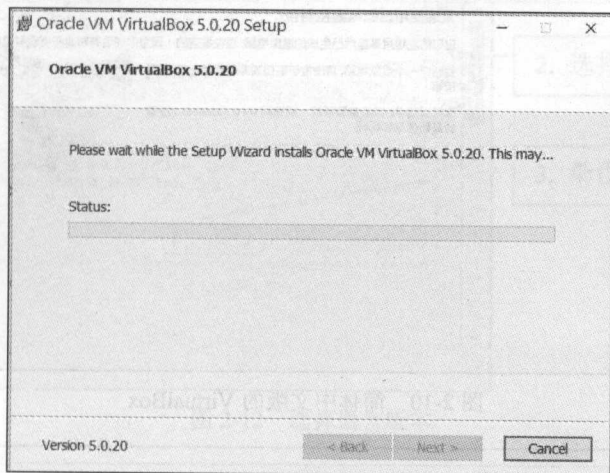


图 2-8 开始进行安装

步骤 09 安装完成

安装完成后单击 Finish 按钮（见图 2-9），就会启动 VirtualBox。



图 2-9 完成安装

步骤 10 启动 VirtualBox 的界面

重新启动后，可以看到 VirtualBox 的屏幕显示界面。安装 VirtualBox 后，默认的语言版本是 Windows 的默认语言版本，也就是说：如果 Windows 安装的是简体中文版，VirtualBox 选择就是简体中文版，如图 2-10 所示。

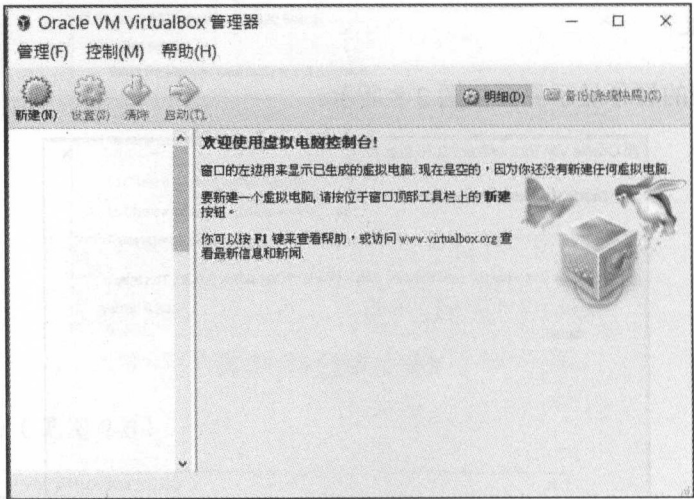


图 2-10 简体中文版的 VirtualBox

2.2 设置 VirtualBox 语言版本

如果安装 VirtualBox 后，默认的语言版本不是你想要的，可以参照下列步骤来设置 VirtualBox 的语言版本。以下范例以英文版为例，设置 VirtualBox 语言版本，操作步骤如下。

步骤 01 选择菜单 File→Preferences...（见图 2-11）

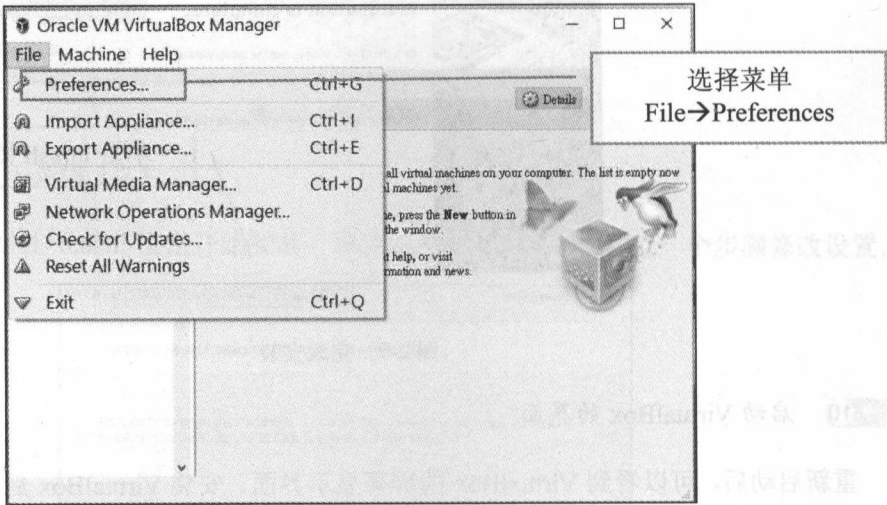


图 2-11 选择 File 菜单下的 Preferences...

步骤 02 选择语言版本

单击菜单后，会打开 Preferences 窗口。在图 2-12 所示的界面中选择语言版本。

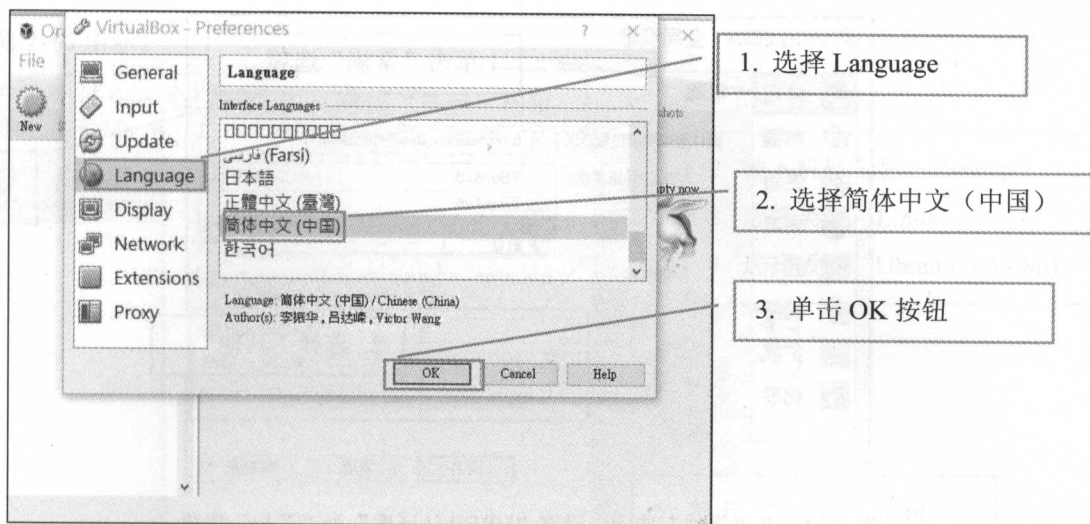


图 2-12 选择语言版本

2.3 设置 VirtualBox 存储文件夹

当创建虚拟主机时，VirtualBox 会创建一个文件，用于存储此虚拟主机的所有数据，默认在 C 盘。但是，因为虚拟主机文件通常很占空间，所以建议设置在空闲空间比较大的磁盘上，例如 D 盘或 E 盘，这样也比较容易进行数据备份。

接下来，我们将介绍如何设置 VirtualBox 默认存储文件夹。

步骤 01 单击菜单选择：“管理” → “全局设定”（见图 2-13）

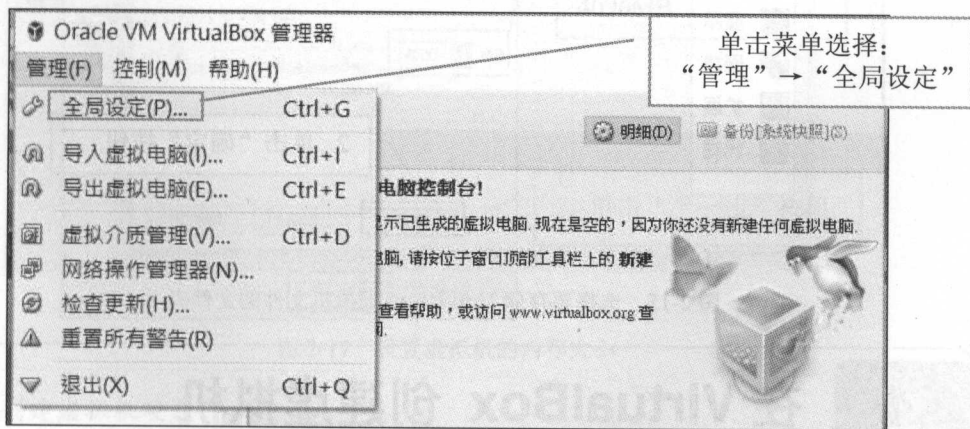


图 2-13 在“管理”菜单中选择“全局设定”

步骤 02 全局设定界面

选择“常规”设置，打开“VRDP 认证库”下拉菜单，从中选择“其他”，如图 2-14 所示。

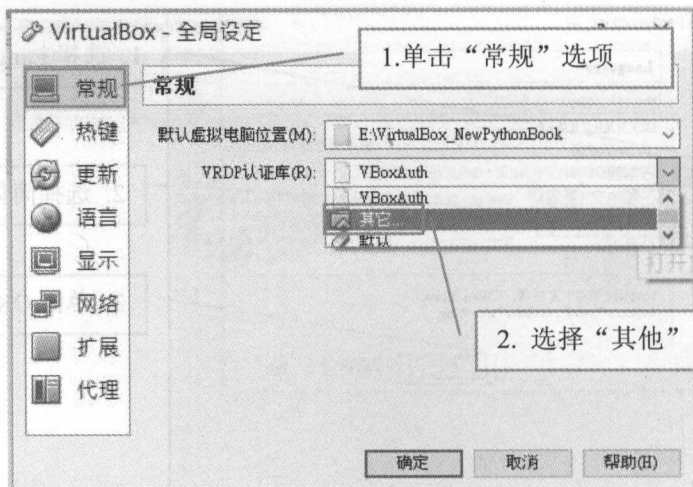


图 2-14 在“常规”选项中设置“VRDP 认证库”为“其他”选项

步骤 03 选择文件夹

选择要存储 VirtualBox 虚拟机文件的文件夹，选好后单击“确定”按钮，如图 2-15 所示。

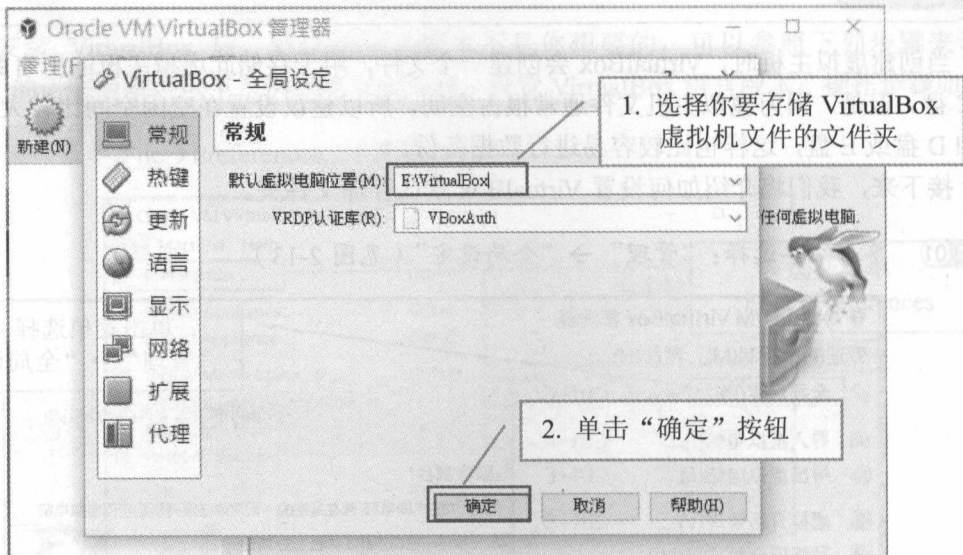


图 2-15 选择要存储 VirtualBox 虚拟机文件的文件夹

2.4 在 VirtualBox 创建虚拟机

接下来，我们要新建一台虚拟机。

步骤 01 创建虚拟机

回到 VirtualBox 启动后的界面，单击“新建”按钮，就会出现“新建虚拟电脑”对话框，

如图 2-16 所示。

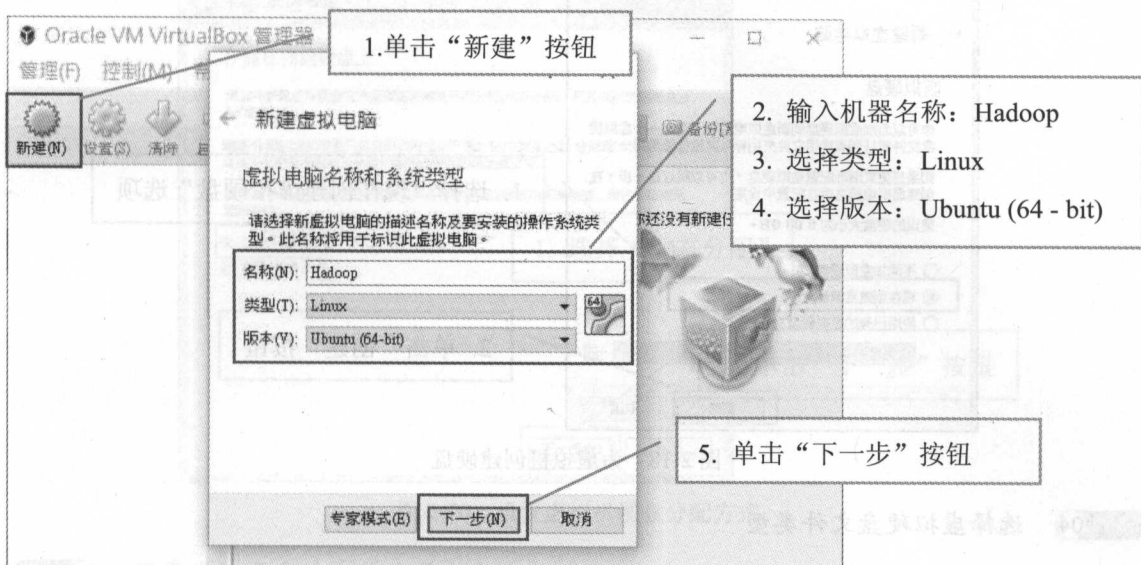


图 2-16 在“新建虚拟电脑”对话框中设置基本信息

步骤 02 设置虚拟机内存的大小 (见图 2-17)

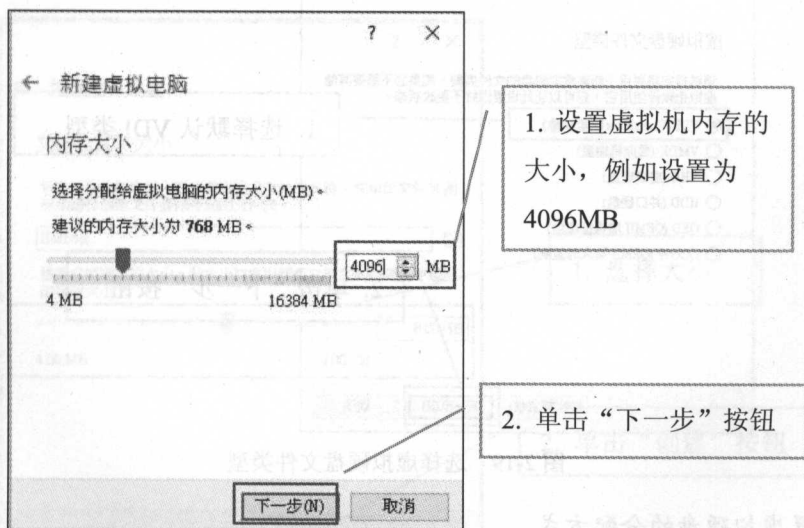


图 2-17 设置虚拟机的内存大小

步骤 03 创建虚拟机硬盘

选择“现在创建虚拟硬盘”，再单击“创建”按钮，如图 2-18 所示。

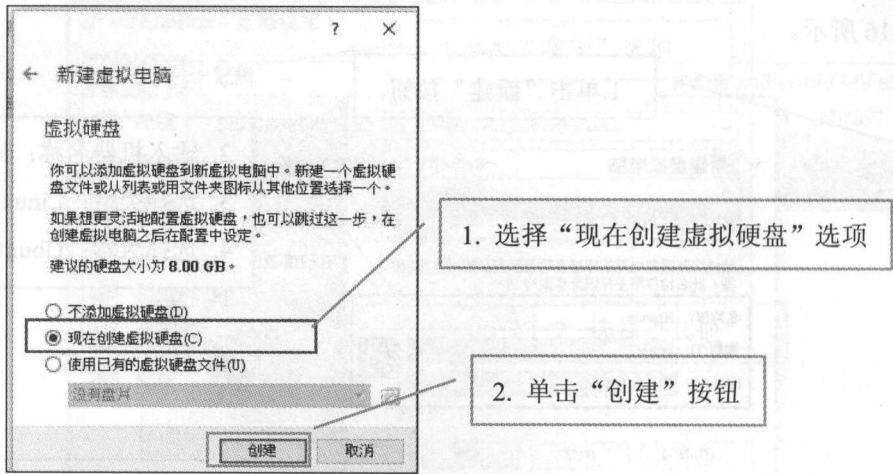


图 2-18 为虚拟机创建硬盘

步骤 04 选择虚拟硬盘文件类型

选择虚拟硬盘的文件格式，默认为 VDI，如图 2-19 所示。

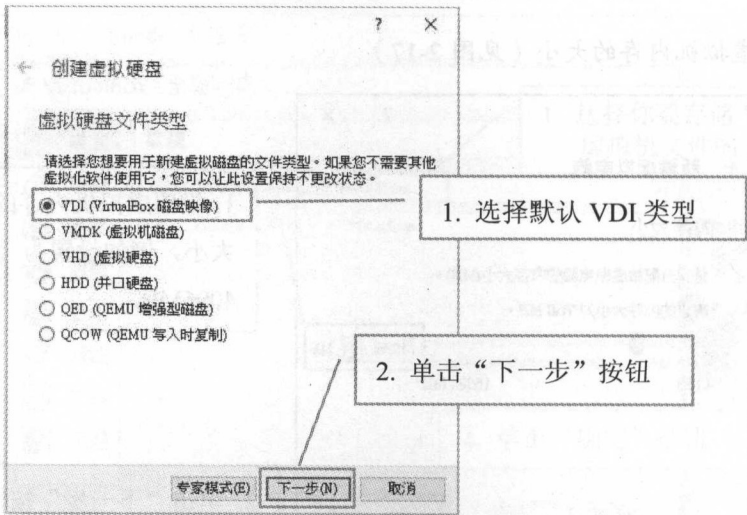


图 2-19 选择虚拟硬盘文件类型

步骤 05 设置虚拟硬盘的分配方式

设置虚拟硬盘的分配方式为“动态分配”，如图 2-20 所示。选择动态分配的好处是不用担心会占用太多硬盘空间。虚拟硬盘会随着虚拟机扩展慢慢地增加存储空间。

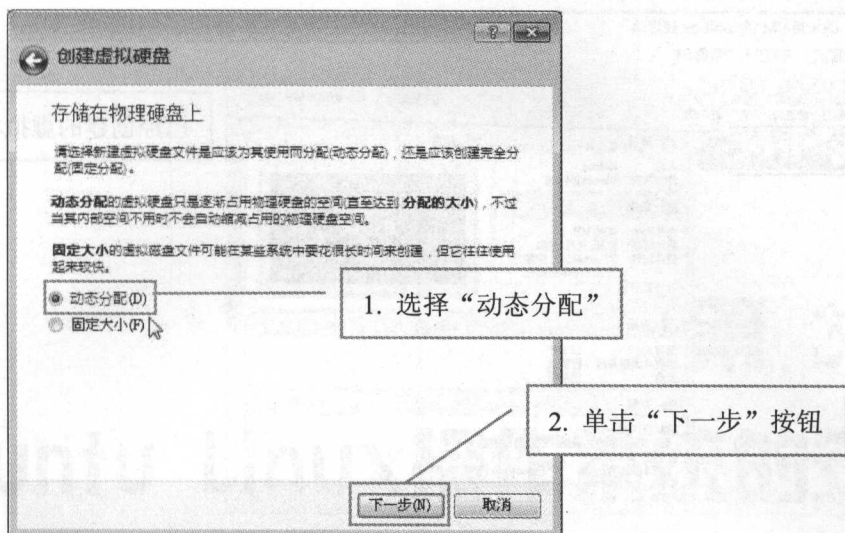


图 2-20 设置虚拟机硬盘分配方式

步骤 06 设置虚拟硬盘的文件位置和大小

接下来，设置虚拟硬盘的文件位置和大小，如图 2-21 所示。文件会创建于之前所设置的默认文件夹中。文件大小设置为 100GB（这是上限值，实际文件会动态增加到上限为止）。

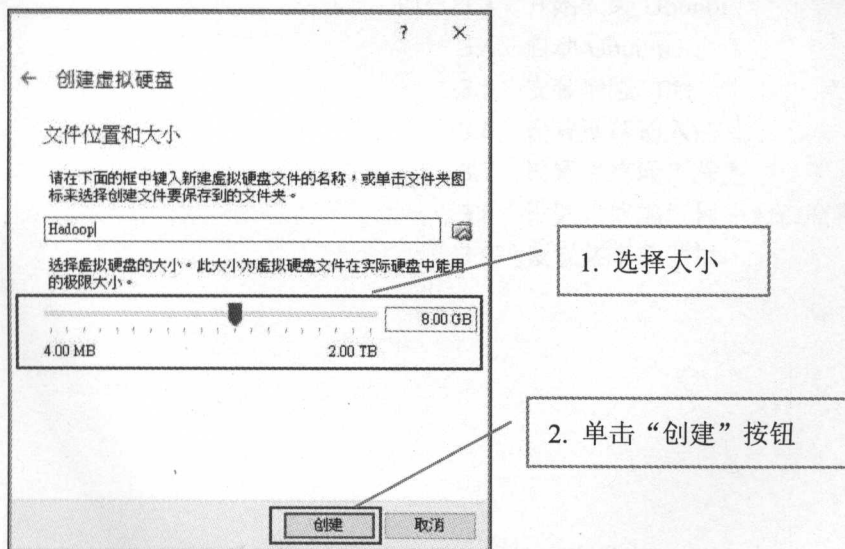


图 2-21 设置虚拟硬盘的文件位置和大小

步骤 07 已经创建的虚拟机

创建完成之后，就可以看到虚拟机的图标了，如图 2-22 所示。

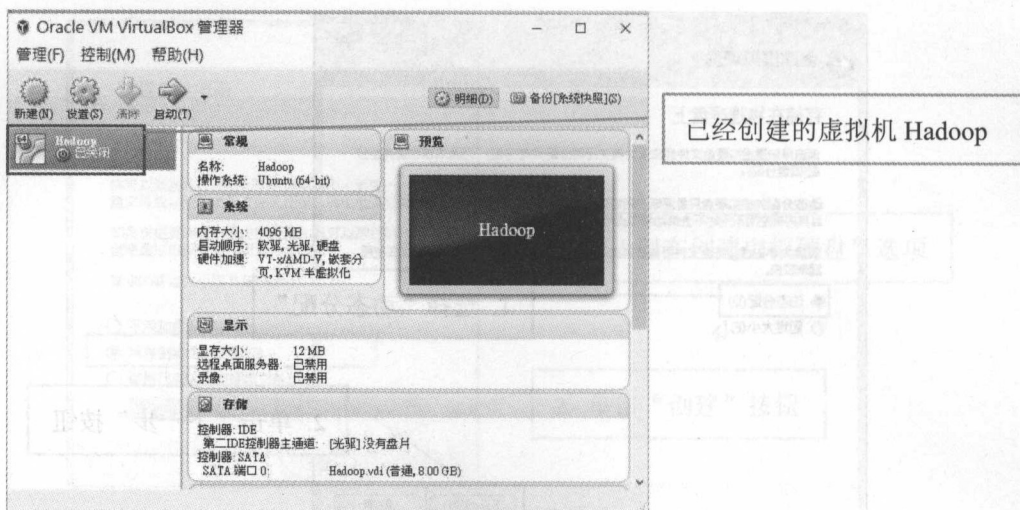
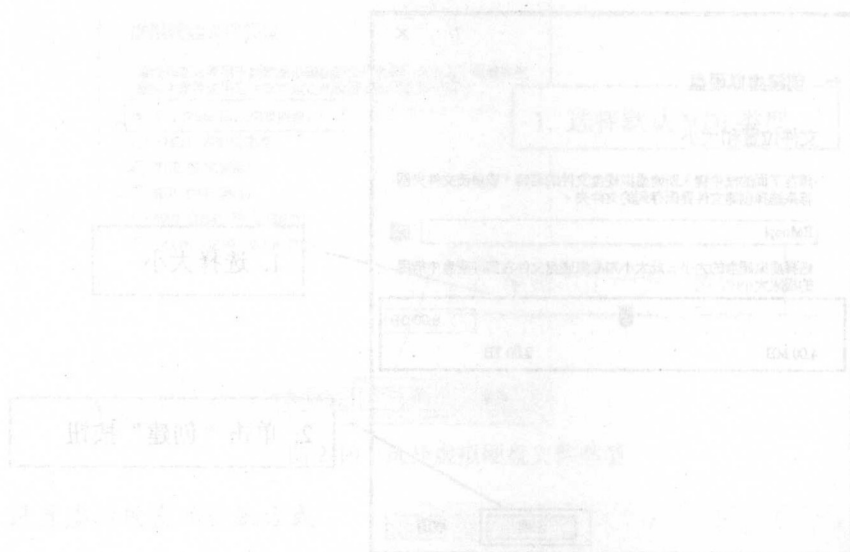


图 2-22 创建好虚拟机



第 3 章

Ubuntu Linux操作系统的安装

- 3.1 下载安装 Ubuntu 的光盘文件
- 3.2 在 Virtual 设置 Ubuntu 虚拟光盘文件
- 3.3 开始安装 Ubuntu
- 3.4 启动 Ubuntu
- 3.5 安装增强功能
- 3.6 设置默认输入法
- 3.7 设置“终端”程序
- 3.8 设置“终端”程序为白底黑字
- 3.9 设置共享剪贴板

Ubuntu 是众多 Linux 操作系统版本中的一种，由 Canonical 公司维护并发行。Ubuntu 来自非洲南部一带，意为“乐于分享”。Ubuntu 提供了 GNOME 桌面环境，是一个开放源码、功能强大而且免费的操作系统，你可以自由下载、复制、使用甚至修改它。除了免费的操作系统本身外，其中还包括文字数据处理、影音播放、图像处理、刻录等各种软件。

3.1 下载安装 Ubuntu 的光盘文件

步骤 01 下载 Ubuntu 的网址

你可以打开浏览器输入 <http://www.ubuntu.com/download/alternative-downloads> 网址，连接至 Ubuntu 官网，下载安装光盘文件，如图 3-1 所示。

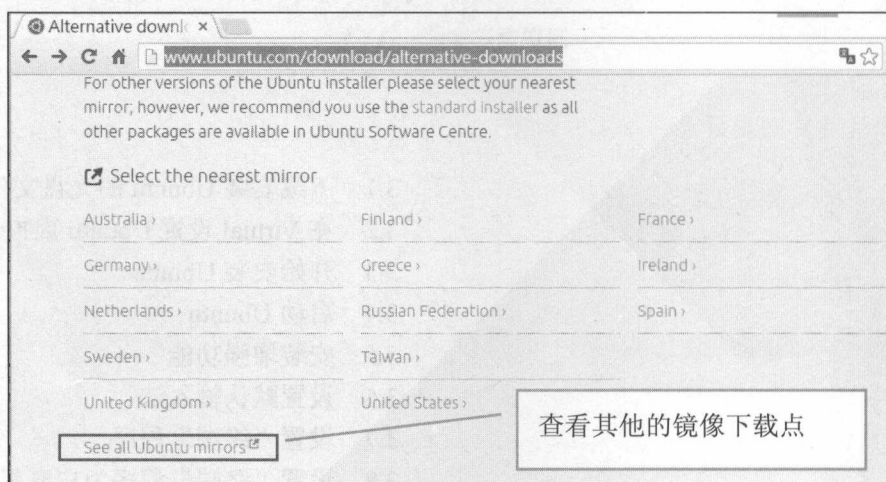


图 3-1 连接至 Ubuntu 官网

步骤 02 选择镜像 (Mirror) 下载点 (见图 3-2)



图 3-2 选择镜像下载点

步骤 03 用鼠标单击要下载 Ubuntu 的网址（见图 3-3）

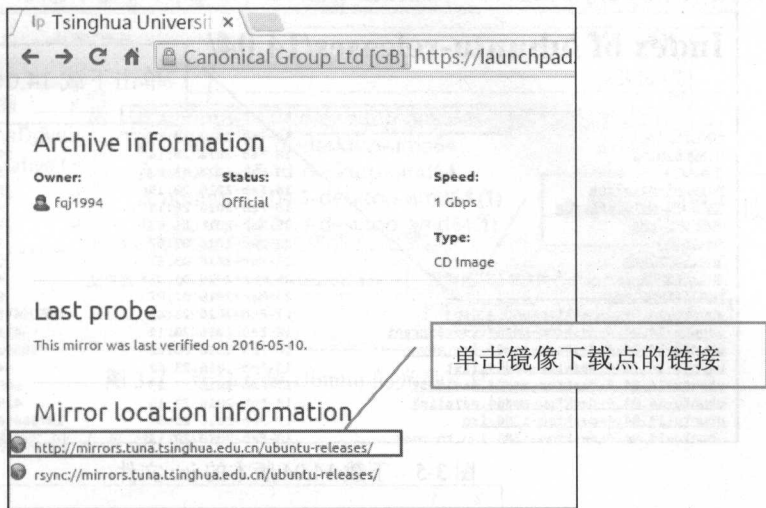


图 3-3 用鼠标单击镜像下载点的链接

步骤 04 用鼠标单击选择 14.04 版本（见图 3-4）

本书选择安装 14.04 版本的 Ubuntu。

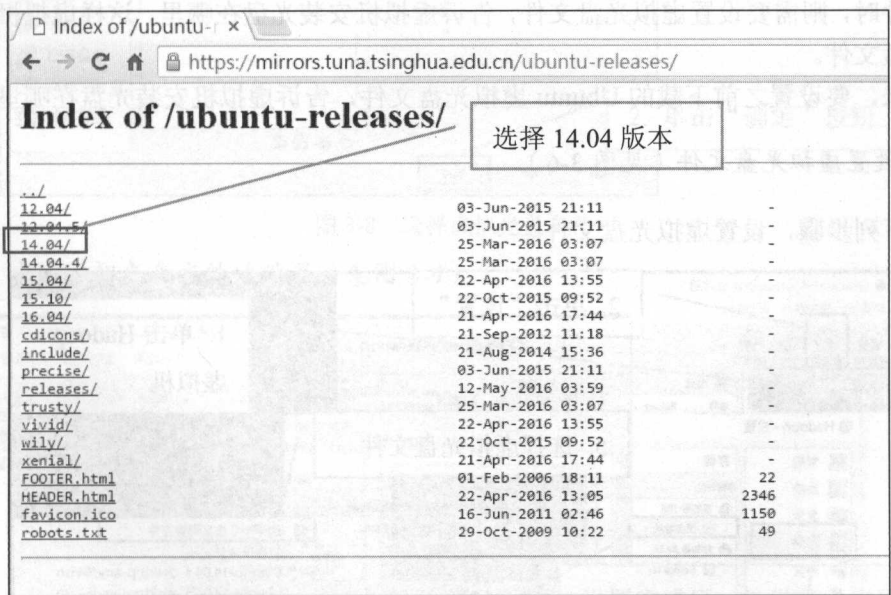


图 3-4 选择 14.04 版本

步骤 05 下载 14.04 版本的 iso 文件（见图 3-5）

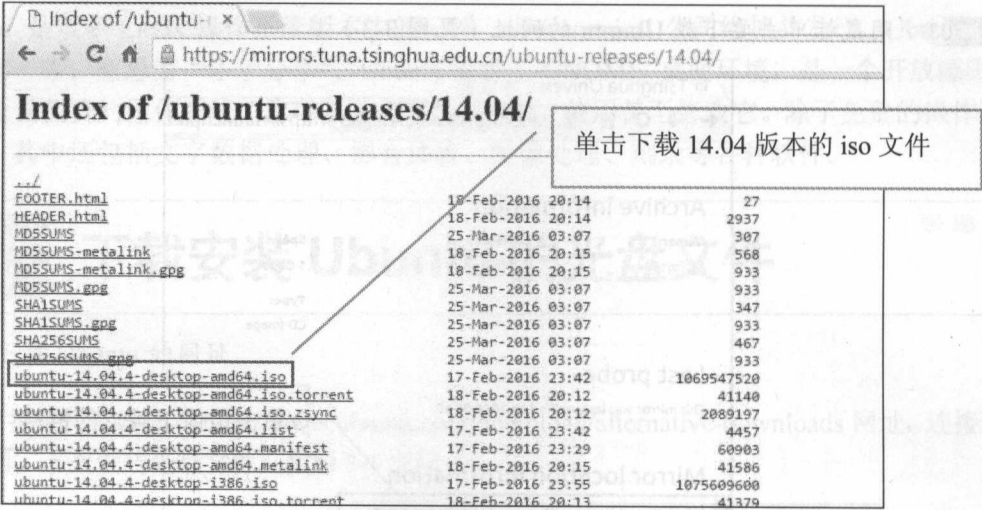


图 3-5 下载 14.04 版本的 iso 文件

3.2 在 Virtual 设置 Ubuntu 虚拟光盘文件

在物理计算机中安装软件时，就是直接把光盘片放到光驱中进行安装。但是，在虚拟机中要安装软件时，则需要设置虚拟光盘文件，告诉虚拟机安装光盘在哪里，这样虚拟机才能读取到安装光盘文件。

接下来，要设置之前下载的 Ubuntu 虚拟光盘文件，告诉虚拟机安装光盘在哪里。

步骤 01 设置虚拟光盘文件（见图 3-6）

按照下列步骤，设置虚拟光盘文件。

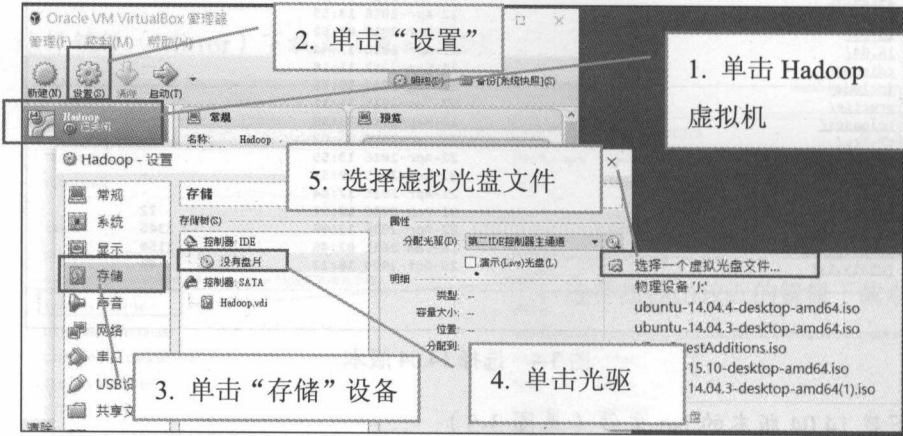


图 3-6 设置虚拟光盘文件

步骤 02 选择 Ubuntu 安装光盘文件（见图 3-7）

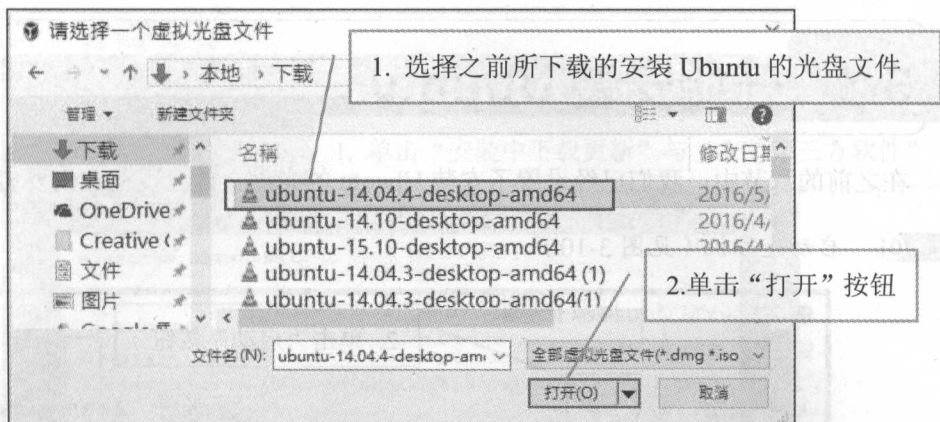


图 3-7 选择安装 Ubuntu 的光盘文件

步骤 03 选择虚拟光盘文件（见图 3-8）

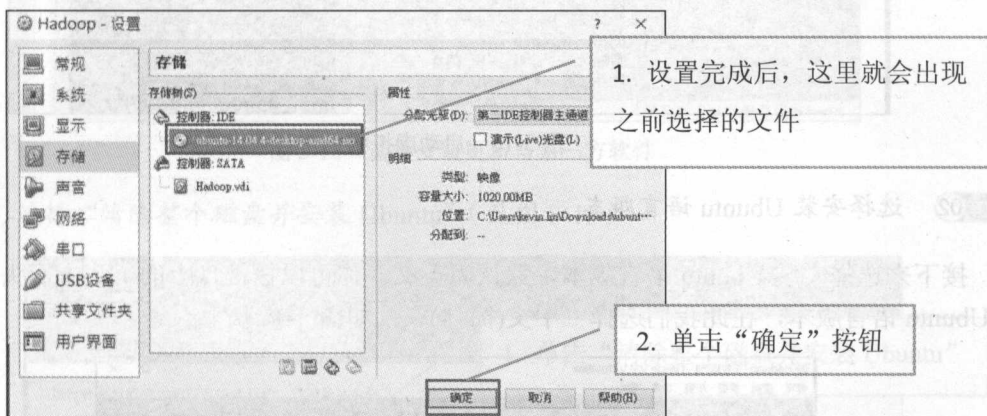


图 3-8 选择虚拟光盘文件

步骤 04 完成虚拟光盘文件的设置（见图 3-9）

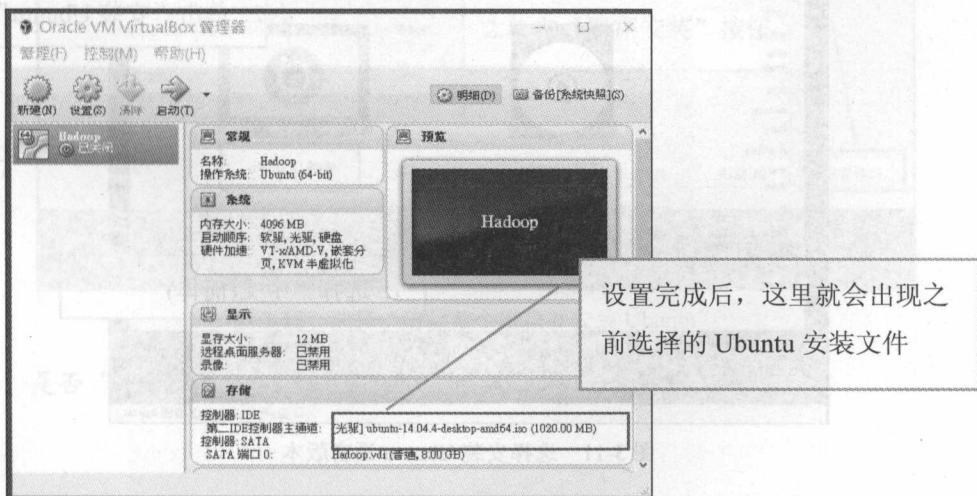


图 3-9 完成虚拟光盘文件的设置

3.3 开始安装 Ubuntu

在之前的章节中，我们已经设置了安装 Ubuntu 的光盘文件。现在我们要启动虚拟机。

步骤 01 启动虚拟机（见图 3-10）

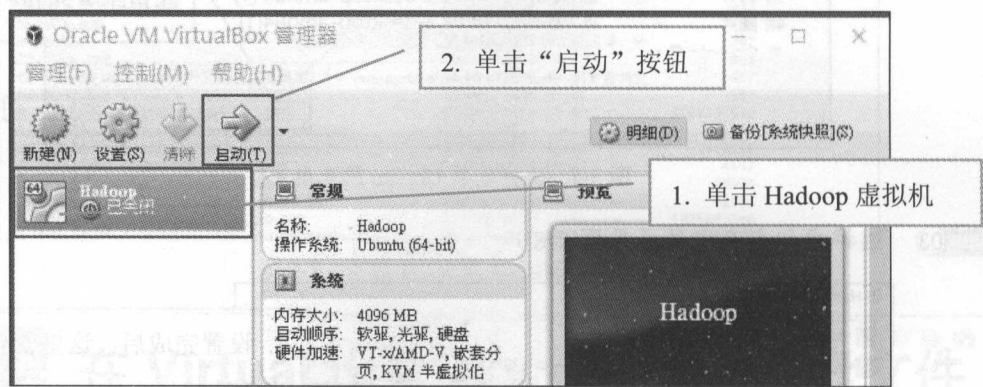


图 3-10 启动虚拟机

步骤 02 选择安装 Ubuntu 语言版本

接下来选择安装 Ubuntu 语言版本，默认为英文。你可以按键盘上的↑、↓箭头键选择安装 Ubuntu 语言版本，在此我们选择“中文(简体)”。如图 3-11 所示。

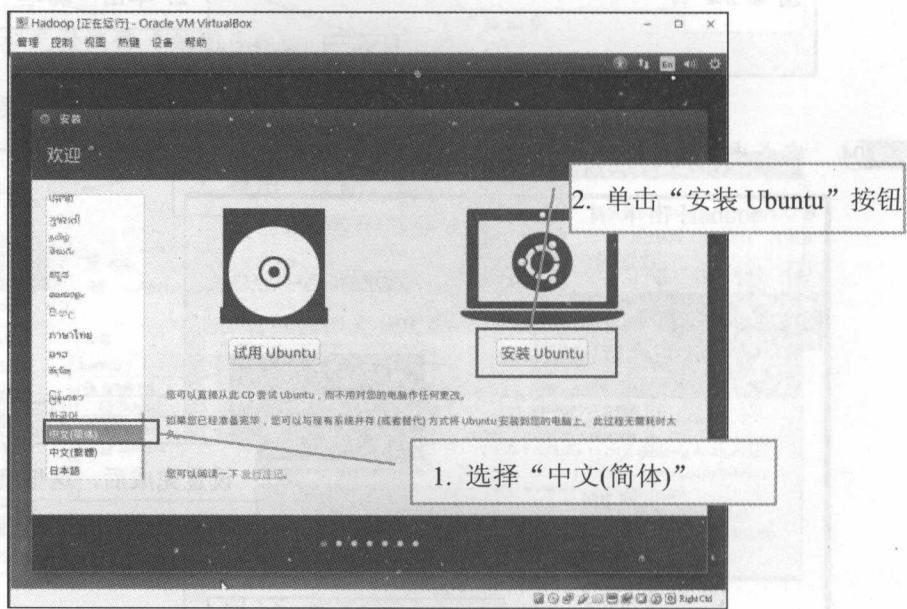


图 3-11 选择安装 Ubuntu 语言版本

步骤 03 选择是否安装更新与安装第三方软件（见图 3-12）

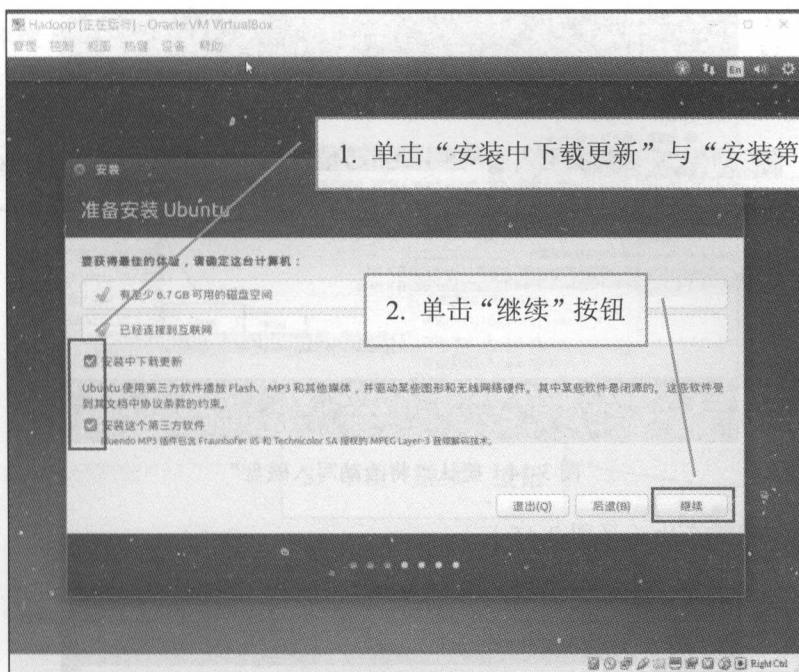


图 3-12 选择安装更新与第三方软件

步骤 04 选择“清除整个磁盘并安装 Ubuntu”（见图 3-13）

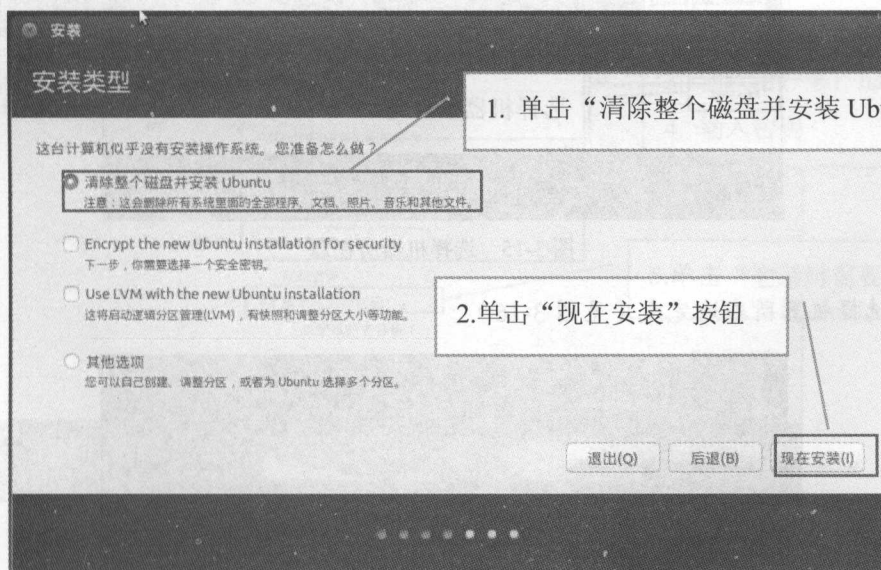


图 3-13 选择“清除整个磁盘并安装 Ubuntu”

步骤 05 是否“将改动写入磁盘”（见图 3-14）

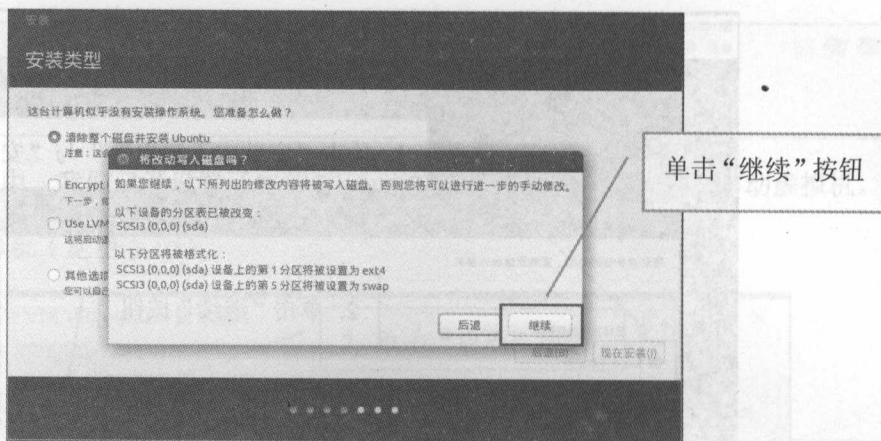


图 3-14 确认“将改动写入磁盘”

步骤 06 选择机器所在地（见图 3-15）

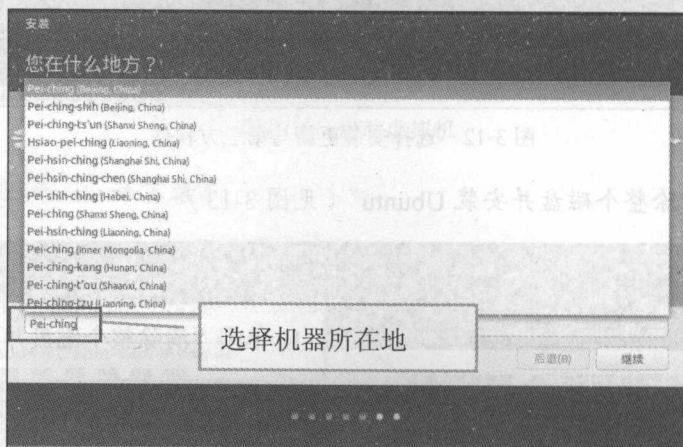


图 3-15 选择机器所在地

步骤 07 选择机器所在地完成（见图 3-16）



图 3-16 完成对机器所在地的选择后继续

步骤 08 选择键盘布局方式 (见图 3-17)

请特别注意键盘布局方式是选择“英语（美国）”而不是选择“汉语（SunPinyin）”。

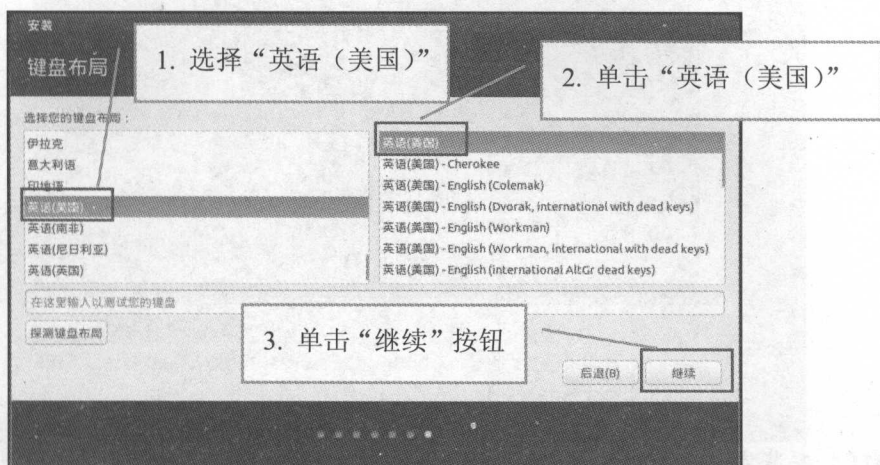


图 3-17 选择键盘布局方式

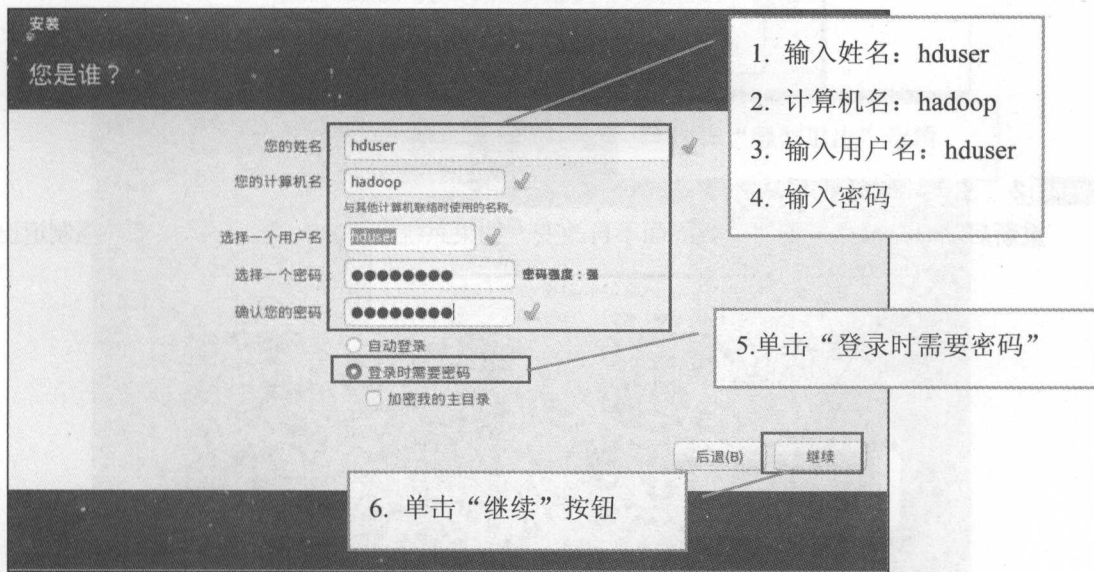
步骤 09 设置姓名、计算机名，选择一个用户名并设置账号密码 (见图 3-18)

图 3-18 设置用户信息

步骤 10 开始安装

开始安装时的屏幕显示界面如图 3-19 所示。

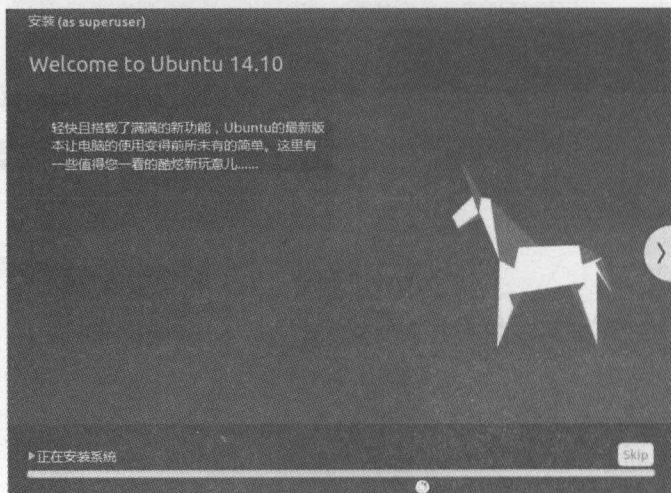


图 3-19 正式开始安装时的显示界面

步骤 11 安装完成并选择重新启动（见图 3-20）

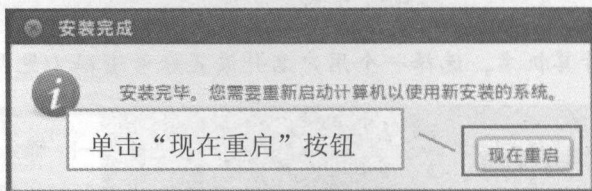


图 3-20 安装完成并选择重新启动

步骤 12 重新启动（见图 3-21）

重新启动时，等候一段时间后界面不再改变。如果系统没有自动退出，可以将其强制退出。

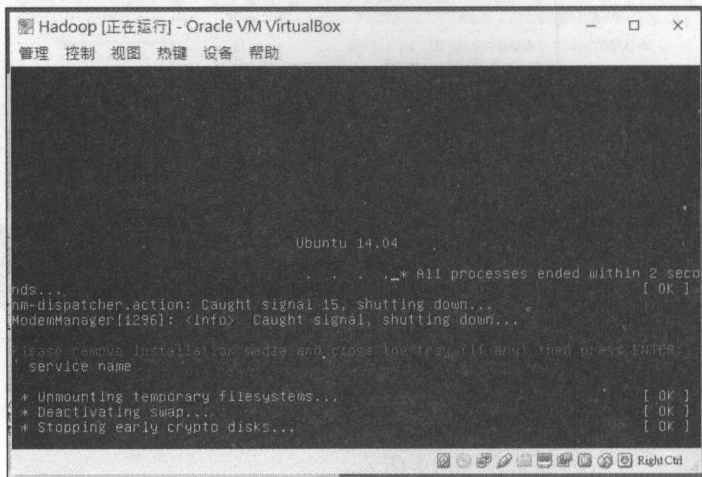


图 3-21 重新启动

步骤 13 强制退出

强制退出的方法如图 3-22 所示。

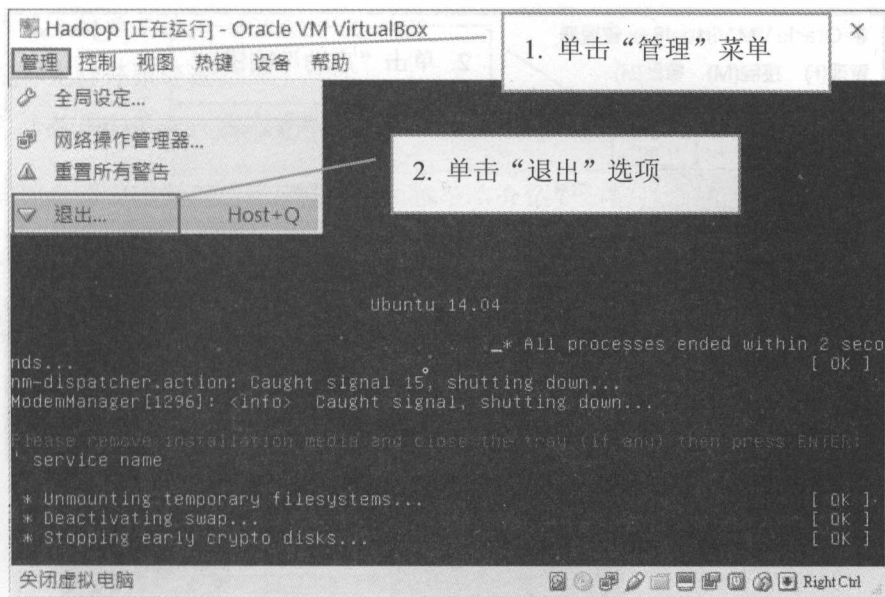


图 3-22 通过“管理”菜单强制关闭虚拟机

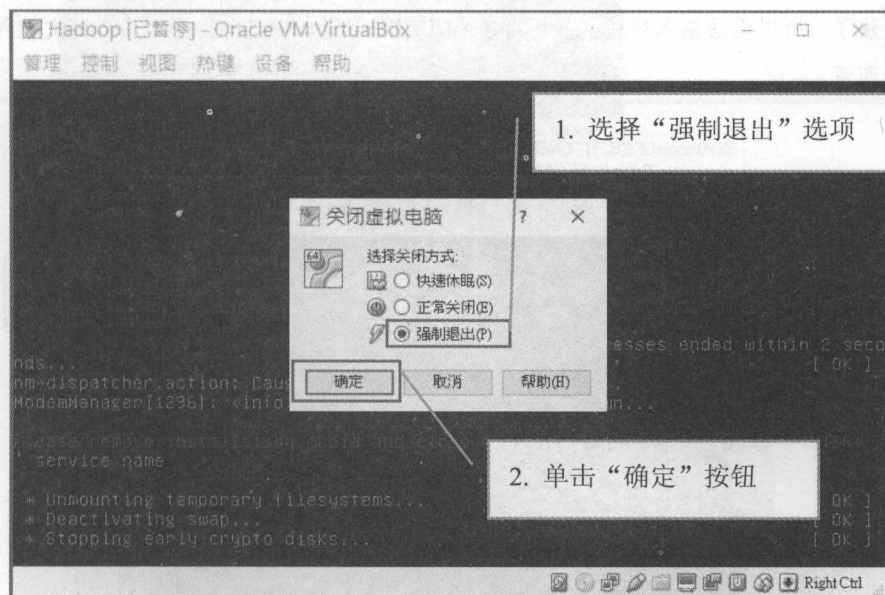
步骤 14 退出（见图 3-23）

图 3-23 在“关闭虚拟电脑”对话框中选择“强制退出”

3.4 启动 Ubuntu

在之前的步骤中，我们已经完成了 Ubuntu 的安装，现在要开机了。

步骤 01 启动 Ubuntu（见图 3-24）

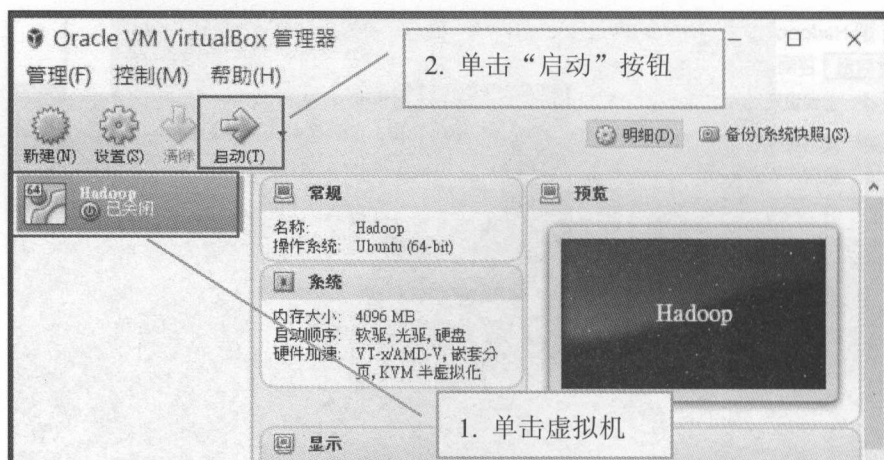


图 3-24 启动 Ubuntu

步骤 02 输入密码 (见图 3-25)

重新启动虚拟机后，系统会要求输入之前设置的密码，然后按 Enter 键即可。

提示

如果输入密码一直不成功，而且输入时文字有下划线，有可能是系统自动切换至中文输入法了，所以无法输入密码。你可以按 Ctrl + Shift 组合键，切换输入法为英文输入，再密码即可。

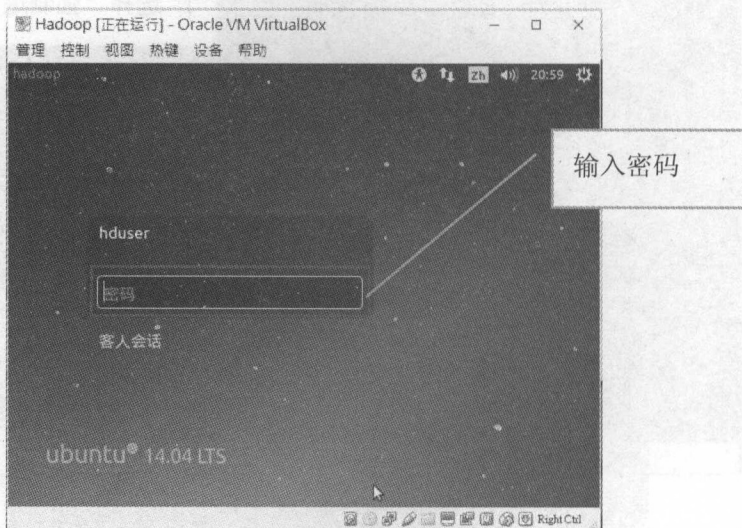


图 3-25 Ubuntu 等待用户输入登录密码

3.5 安装增强功能

Ubuntu 安装基本上已经完成了，只是还是有一些问题：

- 屏幕分辨率不够。
- 鼠标光标停顿延迟。
- 无法与原安装系统共享剪贴板。

这些问题必须安装增强功能才能解决。本节将介绍安装增强功能的步骤。

步骤 01 安装增强功能

单击菜单依次选择“设备”→“安装增强功能”，如图 3-26 所示。

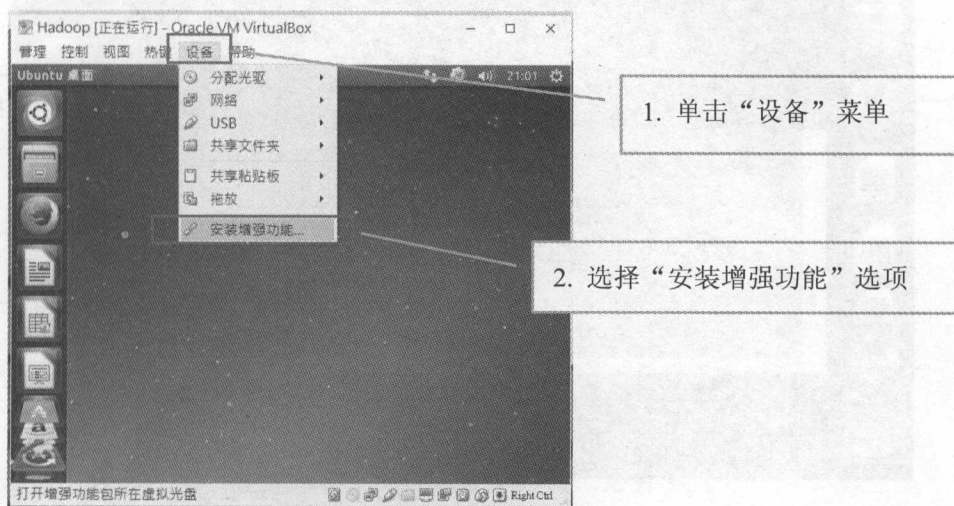


图 3-26 通过菜单选择“安装增强功能”

步骤 02 运行 Guest Additions CD 光盘（见图 3-27）

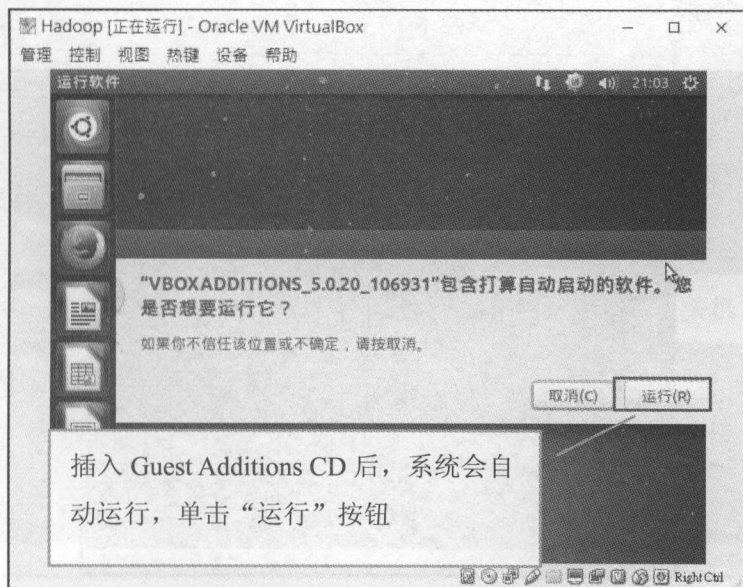


图 3-27 选择运行 Guest Additions CD 光盘

步骤 03 输入超级用户密码

因为安装 Guest Additions CD 光盘程序必须具有超级用户的权限,请输入之前设置的密码,如图 3-28 所示。

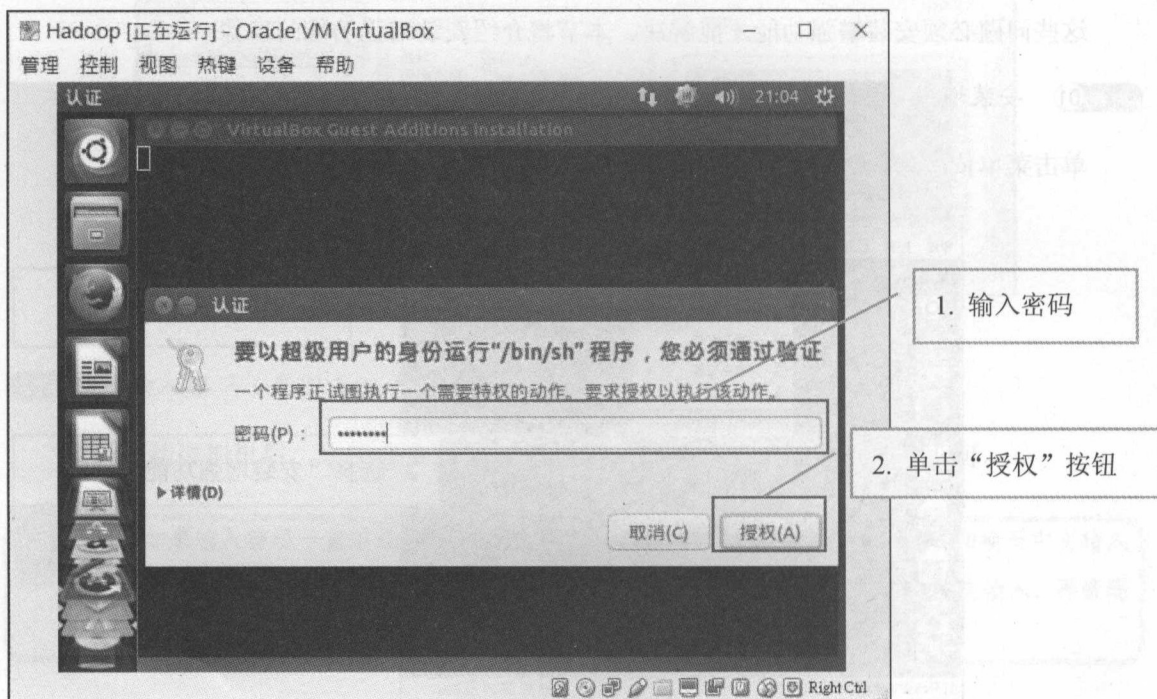


图 3-28 输入超级用户密码

步骤 04 若密码输入成功,则会出现安装 Guest Additions CD 光盘程序的界面,安装完成后如图 3-29 所示。

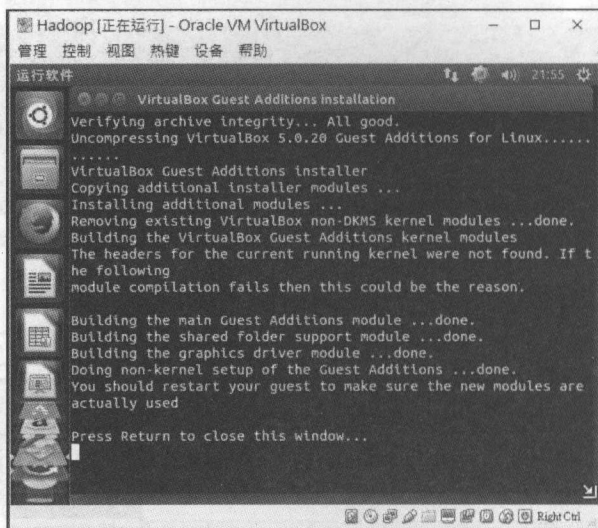


图 3-29 已经完成安装 Guest Additions CD 时的屏幕显示界面

步骤 05 关机（见图 3-30）

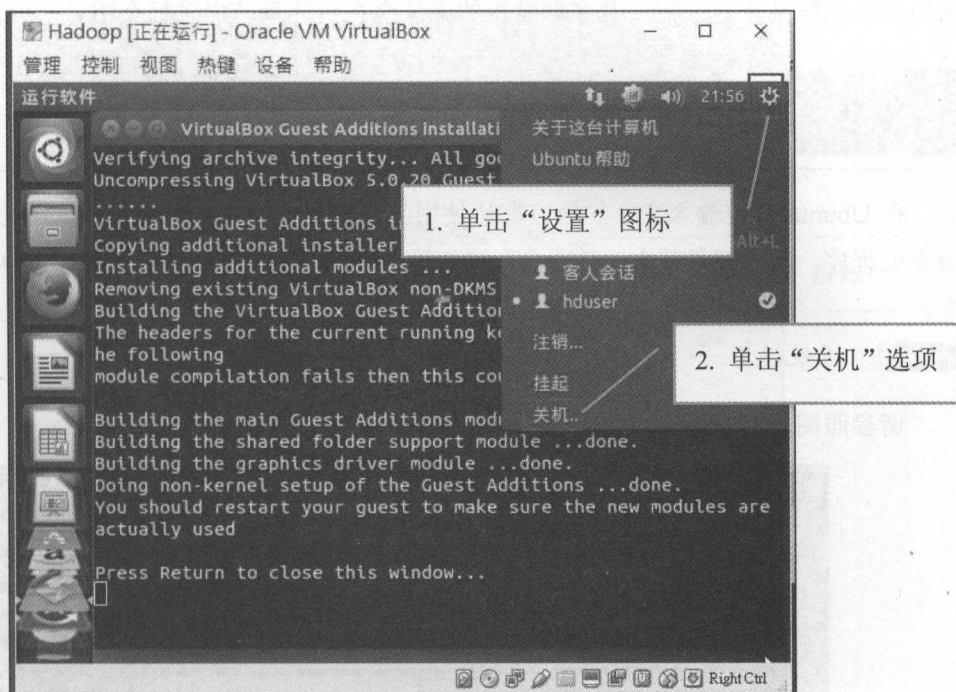


图 3-30 从“设置”工具栏中选择“关机”

步骤 06 最后确认“关机”或“重新启动”的屏幕显示界面（见图 3-31）



图 3-31 最后确认“关机”或“重新启动”的屏幕显示界面

重新启动后，你会发现屏幕分辨率不够、鼠标光标停顿延迟的问题已经解决，但是仍无法与原安装系统共享剪贴板。共享剪贴板的方法会在后续章节中进行介绍。

3.6 设置默认输入法

在 Ubuntu 需要输入文字的地方默认使用的输入法是中文，可是我们大部分情况都是输入命令或程序，这些都是英文，必须自行切换输入法，很不方便。所以最好将默认输入法改为英文。

步骤 01 系统设置

请参照图 3-32 所示的方法打开“系统设置”窗口。

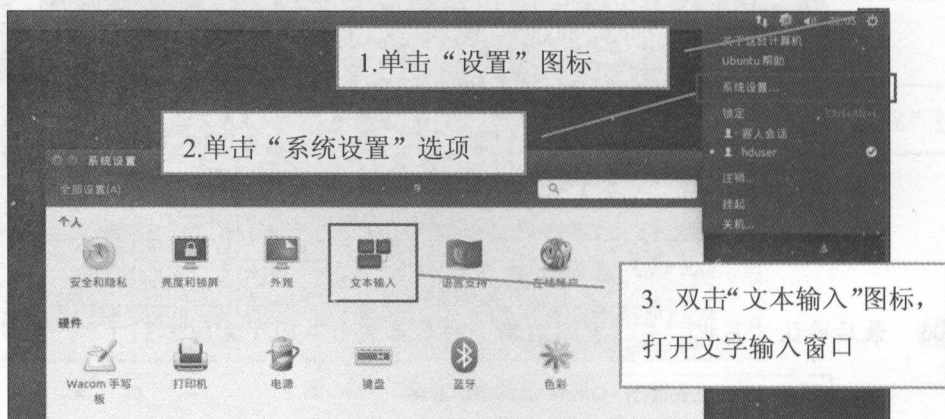


图 3-32 打开“系统设置”窗口

步骤 02 添加输入源（见图 3-33）

打开“文本输入”窗口，默认的文本输入是“汉语(SunPinyin)”。

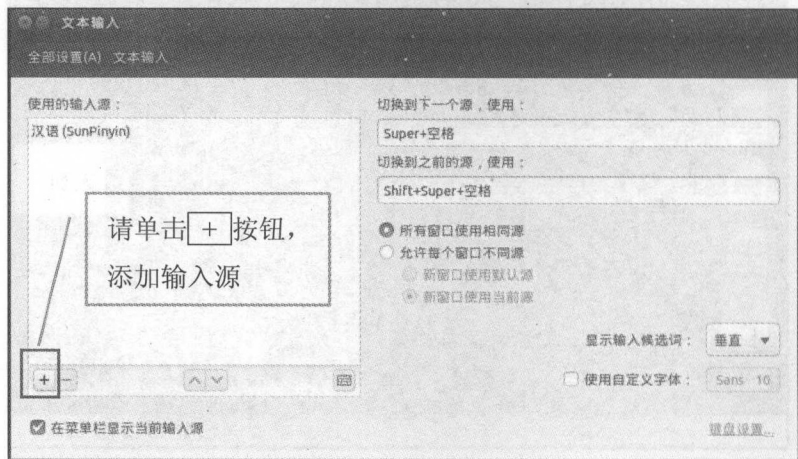


图 3-33 添加输入源

步骤 03 在“选择一个输入源”对话框中选择输入源（见图 3-34）

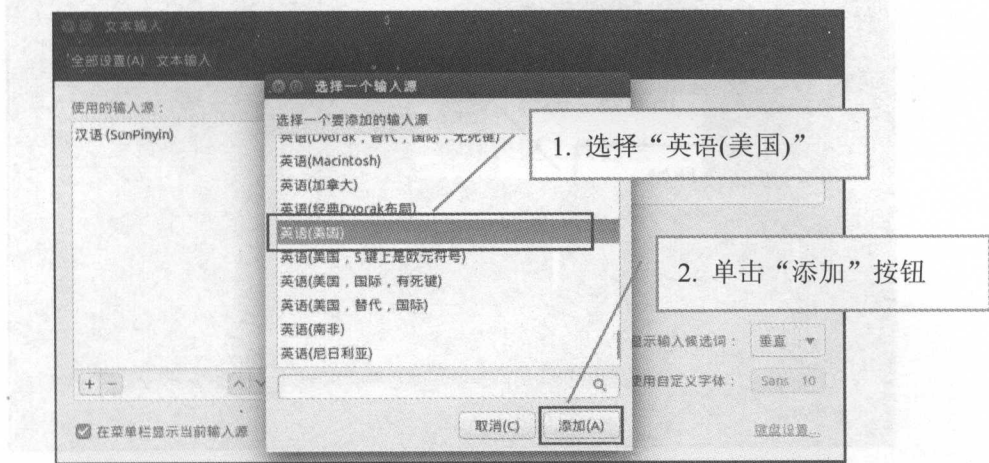


图 3-34 添加“英语(美国)”作为另外一个输入源

步骤 04 调整输入源的顺序（见图 3-35）

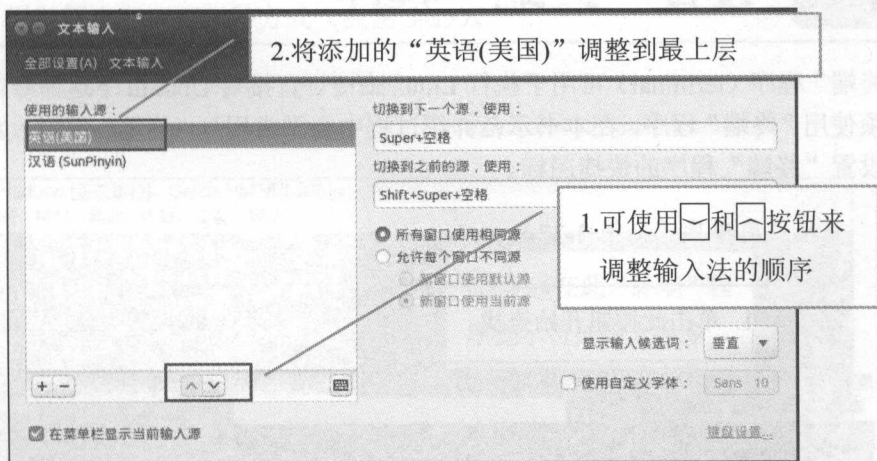


图 3-35 调整输入源的顺序

步骤 05 设置默认快捷键

切换下一个输入法的默认快捷键是 Super + 空格，如图 3-36 所示。

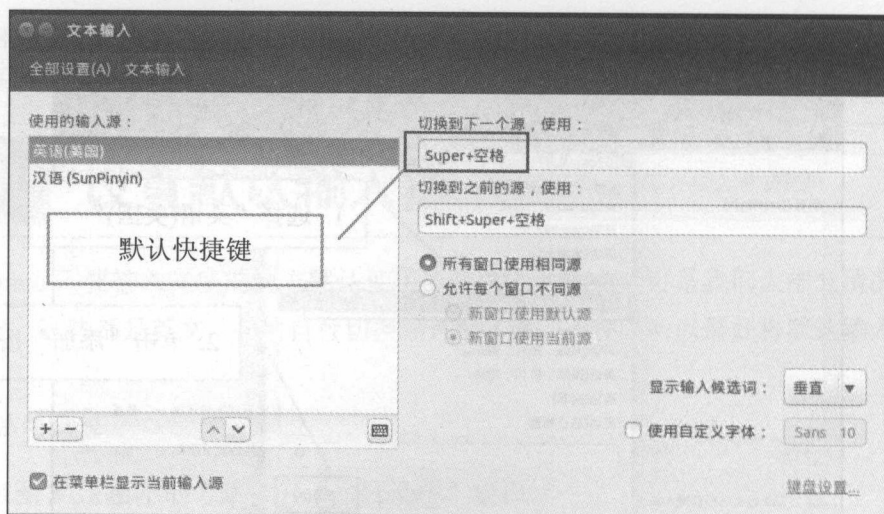


图 3-36 切换下一个输入法的默认快捷键

3.7 设置“终端”程序

“终端”程序 (terminal) 可用于执行 Linux 命令，直接对 Ubuntu 下达命令。安装 Hadoop 时也必须使用“终端”程序，在本书示范介绍过程中会常常用到“终端”程序。为了方便使用，下面先设置“终端”程序的快捷图标。

步骤 01 查找应用程序 (见图 3-37)

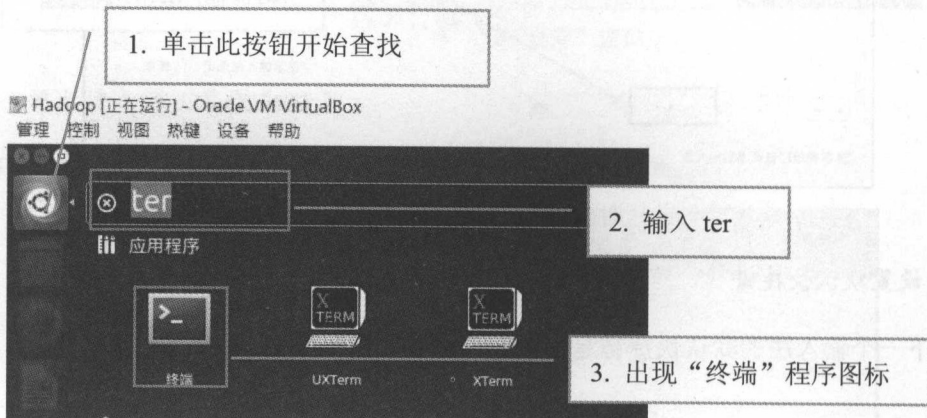


图 3-37 查找应用程序

步骤 02 拖动“终端”程序图标至快捷工具栏 (见图 3-38)

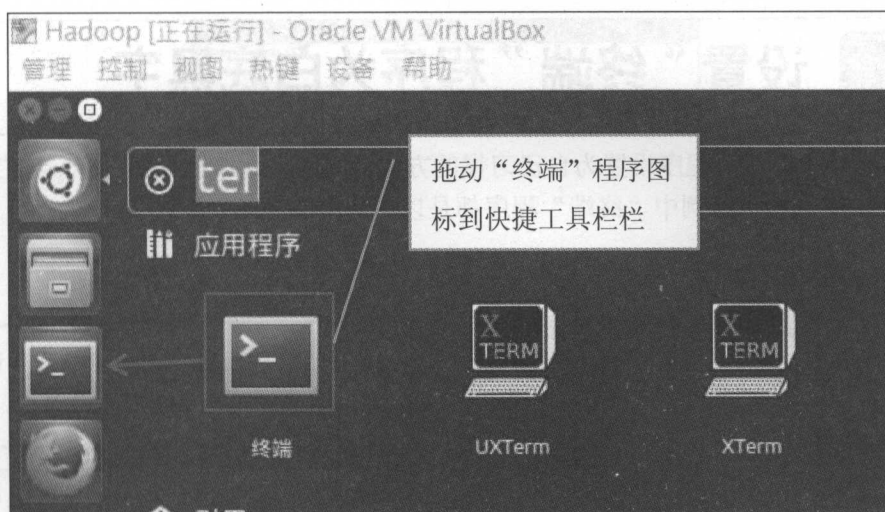


图 3-38 拖动“终端”程序图标到快捷工具栏

步骤 03 启动“终端”程序

启动“终端”程序有以下两种方式（见图 3-39）。

1. 单击快捷工具栏上的“终端”程序图标。
2. 使用快捷键 `Ctrl + Alt + T`。

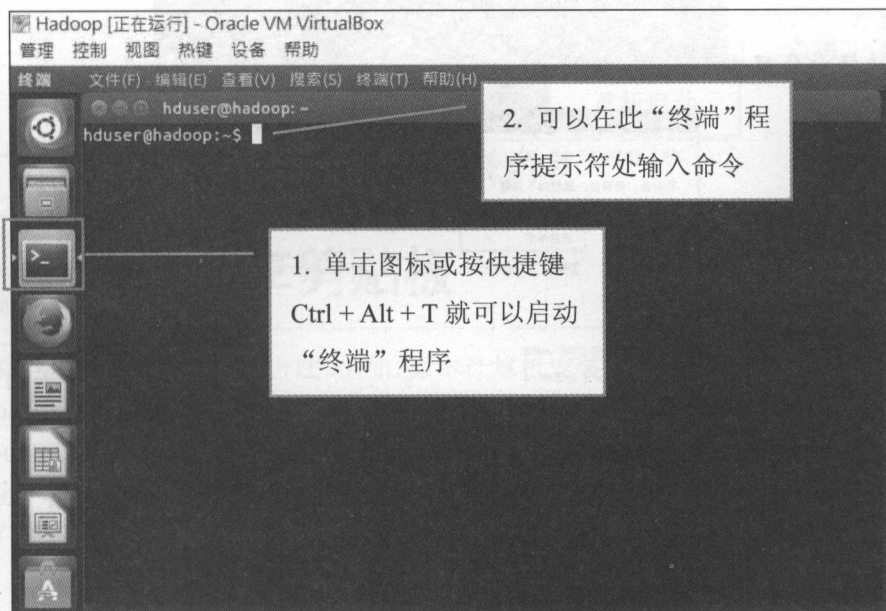


图 3-39 启动“终端”程序

提示

Ubuntu 默认的“终端”程序提示符格式如下：

[当前用户]@[主机]:[当前目录]\$

3.8 设置“终端”程序为白底黑字

用户可以将“终端”程序设置为自己习惯的方式。例如，原本是黑底白字，我们可以将它修改为白底黑字。本书范例中“终端”程序都是以白底黑字显示。

步骤 01 配置文件首选项（见图 3-40）

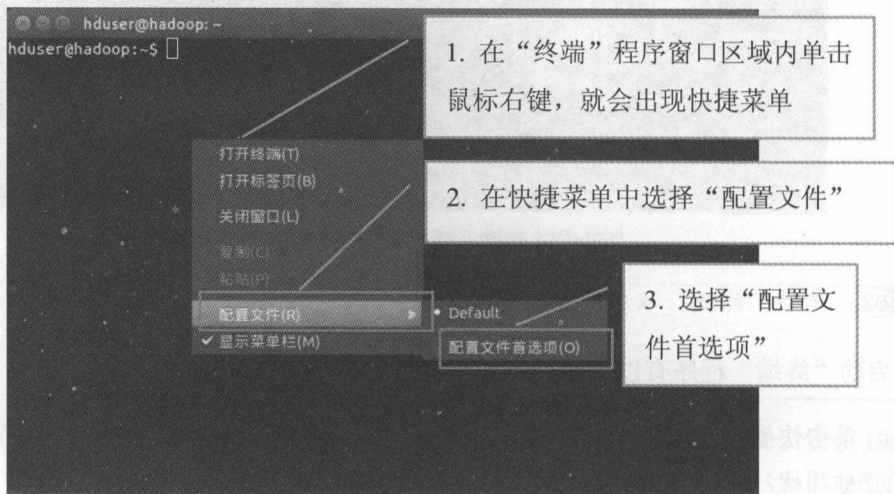


图 3-40 选择“配置文件首选项”

步骤 02 选择白底黑字（见图 3-41）

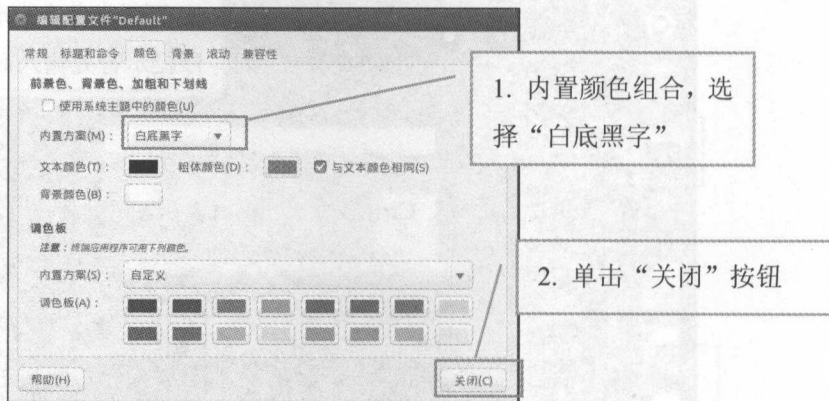


图 3-41 选择白底黑字

步骤 03 已设置“终端”程序的颜色组合

设置完成后，“终端”程序的颜色组合为白底黑字，如图 3-42 所示。

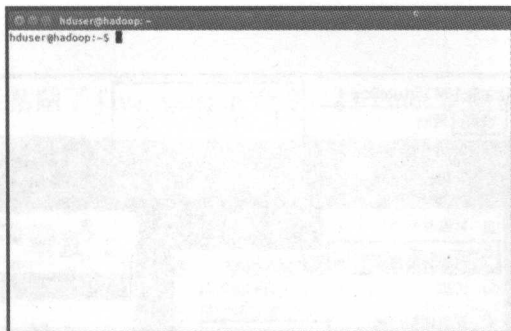


图 3-42 已设置“终端”程序的颜色组合——白底黑字

步骤 04 关机或重新启动（见图 3-43）

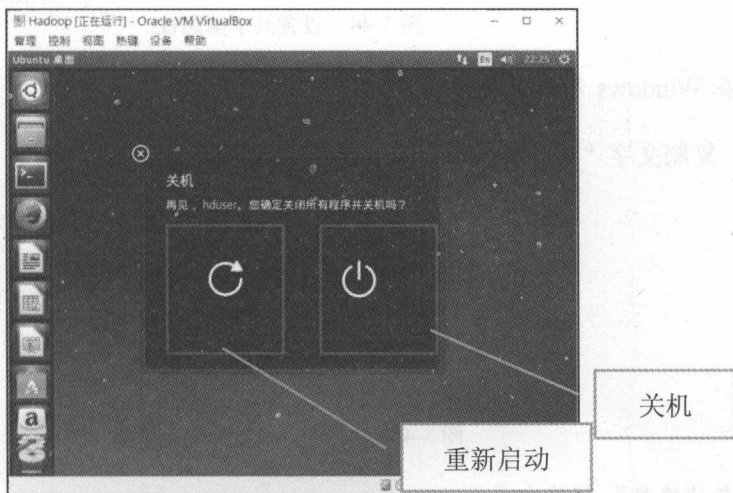


图 3-43 关机或重新启动

3.9 设置共享剪贴板

在此说明共享剪贴板功能的方法。例如，原本计算机安装的是 Windows 7 或者 Windows 10，使用 VirtualBox 安装 Ubuntu，则共享剪贴板功能可以让你在 Windows 系统中复制内容，然后粘贴于 Ubuntu。反之亦然，也可以在 Ubuntu 中单击复制，然后在 Windows 单击粘贴。这个功能很方便，特别是在“终端”程序需要输入命令时，可以使用复制/粘贴节省很多时间，同时也避免了输入错误。

注意，在设置共享剪贴板之前，要参照第 3.5 节的说明先安装增强功能。

步骤 01 设置共享剪贴板

设置共享剪贴板（参考图 3-44）。

注：Windows 操作系统中统一都称为“剪贴板”，但是在 Ubuntu 操作系统的中文版中翻译成了“粘贴板”，在本书中，我们还是统一用“剪贴板”，只是在说明 Ubuntu 插图中使用“粘

贴板”。

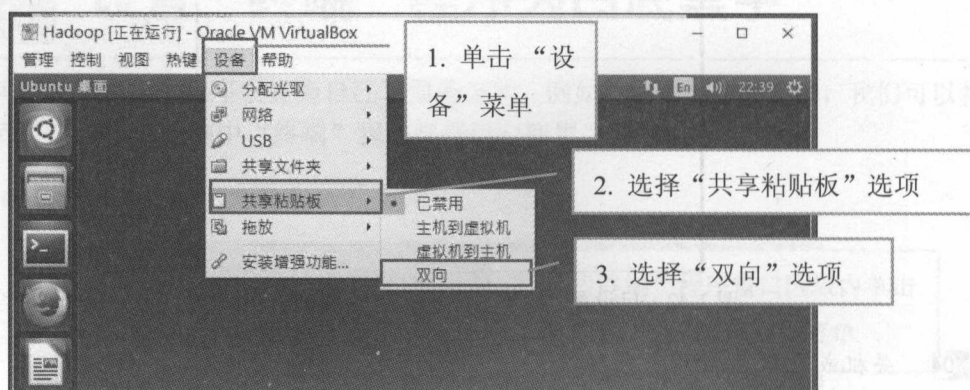


图 3-44 设置共享剪贴板

步骤 02 在 Windows 系统中复制

例如，复制文字“java-version”，如图 3-45 所示。

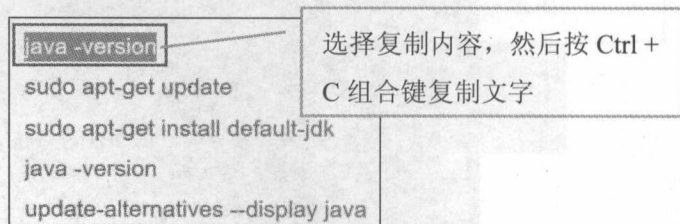


图 3-45 在 Windows 系统中复制

步骤 03 在“终端”程序中粘贴

可以使用下列两种方式粘贴之前复制的文字：在“终端”程序中单击鼠标右键，然后单击“粘贴”选项（见图 3-46）或按 Ctrl + Shift + V 组合键。

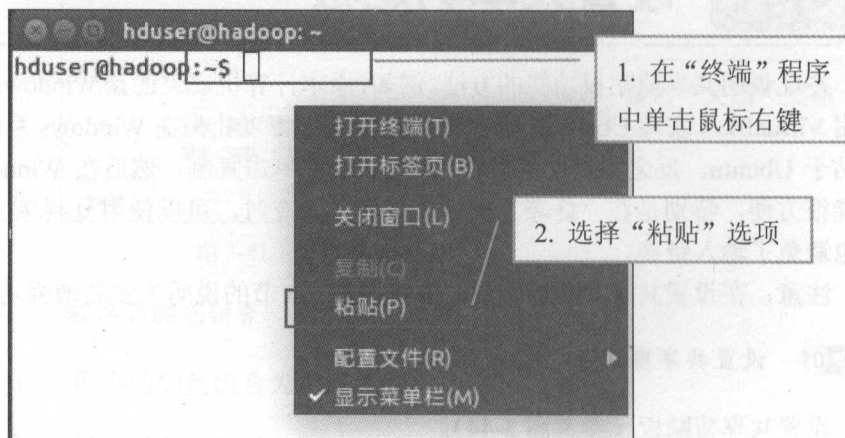
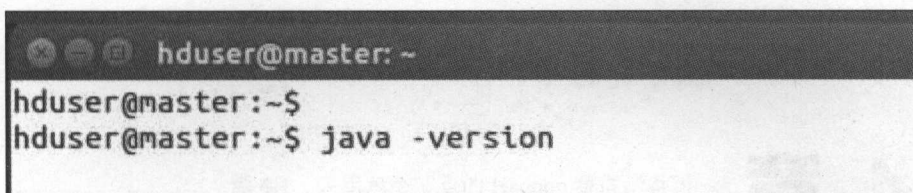


图 3-46 在“终端”程序中粘贴

步骤 04 完成粘贴

如图 3-47 所示，已经粘贴了 `java -version` 命令，按 Enter 键即可开始运行。



```
hduser@master: ~  
hduser@master:~$  
hduser@master:~$ java -version
```

图 3-47 完成粘贴

第 4 章

Hadoop Single Node Cluster的安装

- 4.1 安装 JDK
- 4.2 设置 SSH 无密码登录
- 4.3 下载安装 Hadoop
- 4.4 设置 Hadoop 环境变量
- 4.5 修改 Hadoop 配置设置文件
- 4.6 创建并格式化 HDFS 目录
- 4.7 启动 Hadoop
- 4.8 打开 Hadoop Resource Manager Web 界面
- 4.9 NameNode HDFS Web 界面

Hadoop Single Node Cluster 只以一台机器来建立 Hadoop 环境，你仍然可以使用 Hadoop 命令，只是无法发挥使用多台机器的威力。

因为只有一台服务器，所以所有功能都集中在一台服务器中，如图 4-1 所示。

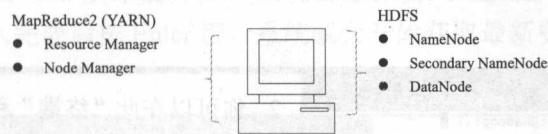


图 4-1 采用单个节点的 Hadoop 集群示意图

安装步骤如下：

顺序	安装步骤	说明
1	安装 JDK	因为 Hadoop 是使用 Java 开发的，所以必须先安装 JDK
2	设置 SSH 无密码登录	Hadoop 必须通过 SSH 与本地计算机以及其他主机连接，所以必须设置 SSH
3	下载安装 Hadoop	到 Hadoop 官网下载 Hadoop 的 64/32 bit 版本（根据操作系统版本来定），并且安装到 Ubuntu 中
4	设置 Hadoop 环境变量	设置每次用户登录时必须设置的环境变量
5	Hadoop 配置文件的设置	在 Hadoop 的/usr/local/hadoop/etc/hadoop 目录下，有很多配置设置文件，通过编辑这些文件来启用基本或是更高级的功能
6	创建并格式化 HDFS 目录	HDFS 目录是存储 HDFS 文件的地方，在启动 Hadoop 之前必须先创建并格式化 HDFS 目录
7	启动 Hadoop	全部设置完成就可以开始启动 Hadoop，并查看 Hadoop 相关进程是否已经启动
8	打开 Hadoop Web 界面	Hadoop 界面可以让你查看当前 Hadoop 的状态：Node 节点、应用程序、任务运行状态

➤ Hadoop 2.6 Single Node Cluster 安装命令

以下安装过程中所需要输入的命令我们已整理在与本书有关的博客文章中。当练习安装时，可以复制博客文章中的命令，然后粘贴到“终端”程序中。这样可以节省打字的时间，也不用担心输出命令（无法在 VirtualBox 虚拟机的 Ubuntu “终端”程序中进行复制和粘贴操作时，请参考第 3.9 节的说明，设置好 VirtualBox 的共享剪贴板）。本书博客的网址为：

<http://blog.sina.com.cn/hadoopsparkbook>

4.1 安装 JDK

因为 Hadoop 是以 Java 开发的，所以必须先安装 Java 环境。

步骤 01 启动“终端”程序

接下来，我们将使用“终端”程序下达命令来安装 Hadoop，所以先启动“终端”程序。你可以单击快捷工具栏的“终端”程序图标，或使用快捷键 Ctrl + Alt + T 启动“终端”程序，如图 4-2 所示。

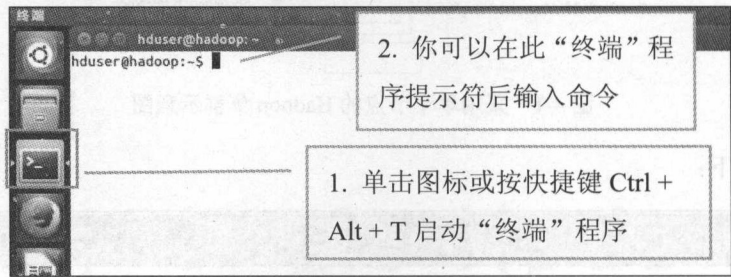


图 4-2 启动“终端”程序

步骤 02 查看当前 Java 版本

在“终端”程序的提示符后输入以下命令并按 Enter 键来运行。

➤ 查看当前 Java 版本

```
Java -version
```

运行结果如图 4-3 所示。

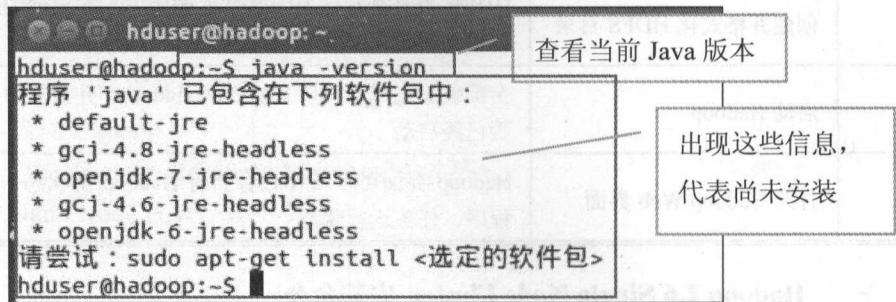


图 4-3 查看当前 Java 版本

步骤 03 sudo apt-get update

在 Linux 中既可使用 apt 进行软件包的管理，也可使用 apt-get 下载安装软件包（或称为套件）。在这里我们会使用 apt-get 安装 jdk。不过，在安装之前，为了获取最新的软件包版本，必须先运行 apt-get update。此命令会连接到 APT Server，更新最新的软件包信息。

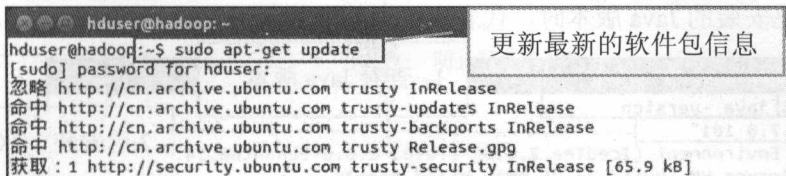
运行 apt-get 必须具有 superuser（超级用户）权限，可是 superuser 权限很大，为了安全性考虑，一般我们在运行时不会以 superuser 来登录系统。我们可以在命令前加上 sudo 命令，系统会询问 superuser 密码（就是安装时输入的密码），这样就可以获得 superuser 权限。

在“终端”程序提示符下输入以下命令。

➤ 连接到 APT Server，更新软件包信息

```
sudo apt-get update
```

运行后显示如图 4-4 所示的界面，系统会让你输入 `hduser` 密码（就是安装时输入的 `superuser` 密码），输入完成后按 `Enter` 键，系统就会开始获取最新更新的软件包、相关文件的对应列表。



```
hduser@hadoop:~$ sudo apt-get update
[sudo] password for hduser:
忽略 http://cn.archive.ubuntu.com trusty InRelease
命中 http://cn.archive.ubuntu.com trusty-updates InRelease
命中 http://cn.archive.ubuntu.com trusty-backports InRelease
命中 http://cn.archive.ubuntu.com trusty Release.gpg
获取 1 http://security.ubuntu.com trusty-security InRelease [65.9 kB]
```

图 4-4 获取最新的软件包信息

步骤 04 Install JDK

在“终端”程序提示符下输入以下命令：

➤ 使用 `apt-get` 安装 JDK

```
sudo apt-get install default-jdk
```

运行后显示如图 4-5 所示的界面，系统会响应：将要占用 39MB 存储空间，询问是否要继续，若要进行，请输入 `Y` 再按 `Enter` 键。



```
hduser@hadoop:~$ sudo apt-get install default-jdk
正在读取软件包列表... 完成
正在分析软件包的依赖关系树
正在读取状态信息... 完成
将会安装下列额外的软件包：
ca-certificates-java default-jre default-jre-headless fonts-dejavu-extra
java-common libatk-wrapper-java libatk-wrapper-java-jni libbonobo2-0
libbonobo2-common libgconf2-4 libgif4 libgnome2-0 libgnome2-bin
libgnome2-common libgnomevfs2-0 libgnomevfs2-common libice-dev libidl-common
libidl0 liborbit-2-0 liborbit2 libpthread-stubs0-dev libsctp1 libsm-dev
libx11-dev libx11-doc libxau-dev libxcb1-dev libxdmcp-dev libxt-dev
lksctp-tools openjdk-7-jdk openjdk-7-jre openjdk-7-jre-headless tzdata
tzdata-java x11proto-core-dev x11proto-input-dev x11proto-kb-dev
xorg-sgml-doctools xtrans-dev
建议安装的软件包：
equivs libbonobo2-bin desktop-base libgnomevfs2-bin libgnomevfs2-extra gamln
fam gnome-mime-data libice-doc libsm-doc libxcb-doc libxt-doc openjdk-7-demo
openjdk-7-source visualvm icedtea-7-plugin icedtea-7-jre-jamvm
sun-java6-fonts fonts-ipafont-gothic fonts-ipafont-mincho ttf-wqy-microhei
ttf-wqy-zenhei ttf-telugu-fonts ttf-orlita-fonts ttf-kannada-fonts
ttf-bengali-fonts
下列【新】软件包将被安装：
ca-certificates-java default-jdk default-jre default-jre-headless
fonts-dejavu-extra java-common libatk-wrapper-java libatk-wrapper-java-jni
libbonobo2-0 libbonobo2-common libgconf2-4 libgif4 libgnome2-0 libgnome2-bin
libgnome2-common libgnomevfs2-0 libgnomevfs2-common libice-dev libidl-common
libidl0 liborbit-2-0 liborbit2 libpthread-stubs0-dev libsctp1 libsm-dev
libx11-dev libx11-doc libxau-dev libxcb1-dev libxdmcp-dev libxt-dev
lksctp-tools openjdk-7-jdk openjdk-7-jre openjdk-7-jre-headless tzdata-java
x11proto-core-dev x11proto-input-dev x11proto-kb-dev xorg-sgml-doctools
xtrans-dev
下列软件包将被升级：
tzdata
升级了 1 个软件包，新安装了 41 个软件包，要卸载 0 个软件包，有 107 个软件包未被
升级。
需要下载 62.3 MB/62.4 MB 的软件包，
解压后会消耗掉 110 MB 的额外空间
您希望继续执行吗？ [Y/n] Y
```

图 4-5 安装 JDK

步骤 05 再次查询 Java 版本

再一次在“终端”程序的提示符下输入以下命令：

➤ 查询 Java 版本

```
Java -version
```

系统响应已安装的 Java 版本时，代表已经成功安装了 JDK，如图 4-6 所示。

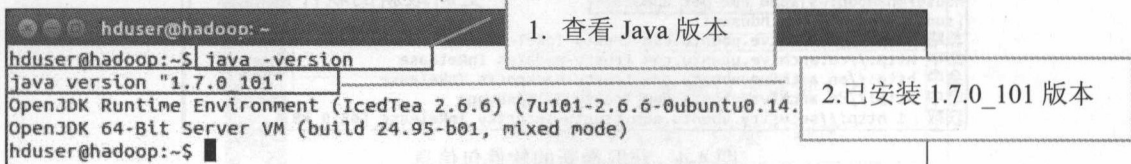


图 4-6 再次查询 Java 版本

步骤 06 查询 Java 安装的位置

你可能会想知道 Java 安装在哪里，可以使用下列命令：

➤ 查询 Java 安装路径

```
update-alternatives --display Java
```

系统会响应安装的路径，如图 4-7 所示。

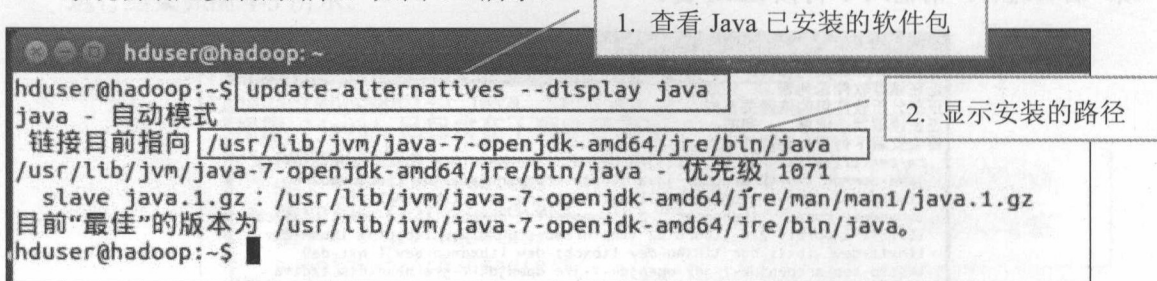


图 4-7 查询 Java 安装的位置

从图 4-7 可以得知，Java 安装在 `/usr/lib/jvm/java-7-openjdk-amd64` 文件夹中。稍后我们会在 `~/.bashrc` 文件中设置此路径。

4.2 设置 SSH 无密码登录

Hadoop 是由很多台服务器所组成的。当我们启动 Hadoop 系统时，NameNode 必须与 DataNode 连接，并管理这些节点（DataNode）。此时系统会要求用户输入密码。为了让系统顺利运行而不需手动输入密码，就需要 SSH 设置成无密码登录。注意，无密码登录并非不需要密码，而是以事先交换的 SSH Key（密钥）来进行身份验证。

如图 4-8 所示，Hadoop 使用 SSH（Secure Shell）连接。这是目前较可靠、专为远程登录

其他服务器提供的安全性协议。通过 SSH 会对所有传输的数据进行加密。利用 SSH 协议可以防止远程管理系统时信息外泄的问题。

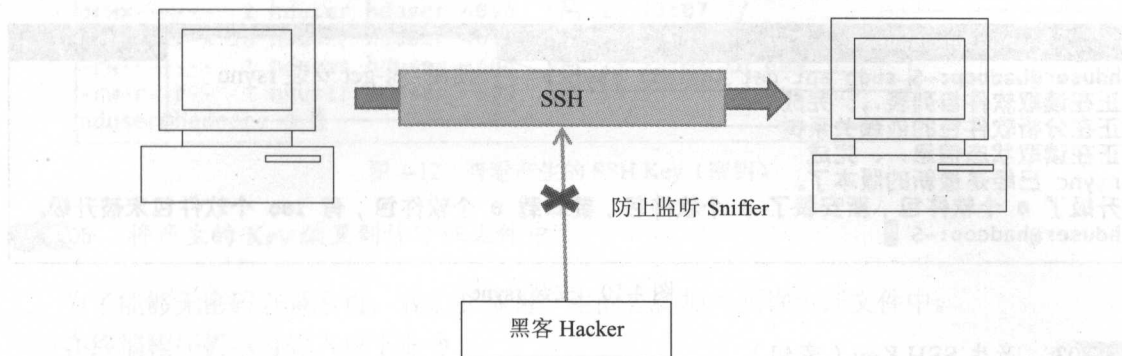


图 4-8 Hadoop 使用 SSH 连接

步骤 01 安装 SSH

在“终端”程序提示符下输入以下命令：

➤ 安装 SSH

```
sudo apt-get install ssh
```

执行后显示如图 4-9 所示的界面，系统会询问是否要继续，若继续，请输入“Y”再按 Enter 键。

```

hduser@hadoop: ~
hduser@hadoop:~$ sudo apt-get install ssh
[sudo] password for hduser:
正在读取软件包列表... 完成
正在分析软件包的依赖关系树
正在读取状态信息... 完成
将会安装下列额外的软件包：
  libck-connector0 ncurses-term openssh-client openssh-server
  openssh-sftp-server ssh-import-id
建议安装的软件包：
  libpam-ssh keychain monkeysphere rssh molly-guard
下列【新】软件包将被安装：
  libck-connector0 ncurses-term openssh-server openssh-sftp-server ssh
  ssh-import-id
下列软件包将被升级：
  openssh-client
升级了 1 个软件包，新安装了 6 个软件包，要卸载 0 个软件包，有 106 个软件包未被升级。
需要下载 620 kB/1,183 kB 的软件包。
解压缩后会消耗掉 3,450 kB 的额外空间。
您希望继续执行吗？ [Y/n]
  
```

图 4-9 安装 SSH

步骤 02 安装 rsync

在“终端”程序提示符下输入以下命令：

➤ 安装 rsync

```
sudo apt-get install rsync
```

运行后界面如图 4-10 所示。

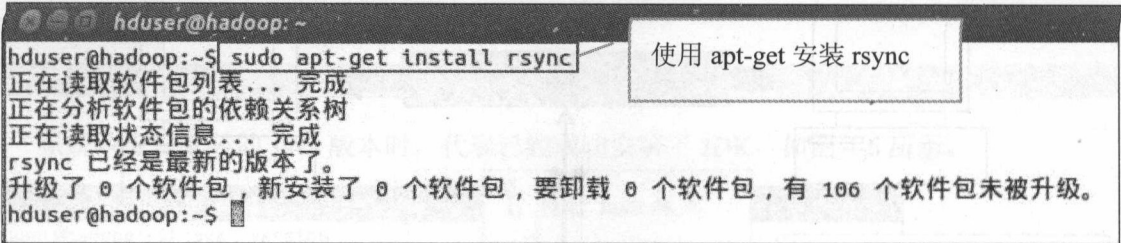


图 4-10 安装 rsync

步骤 03 产生 SSH Key（密钥）

在“终端”程序提示符下输入以下命令：

➤ 产生 SSH Key（密钥）进行后续身份验证

```
ssh-keygen -t dsa -P '' -f ~/.ssh/id_dsa
```

执行后界面如图 4-11 所示。

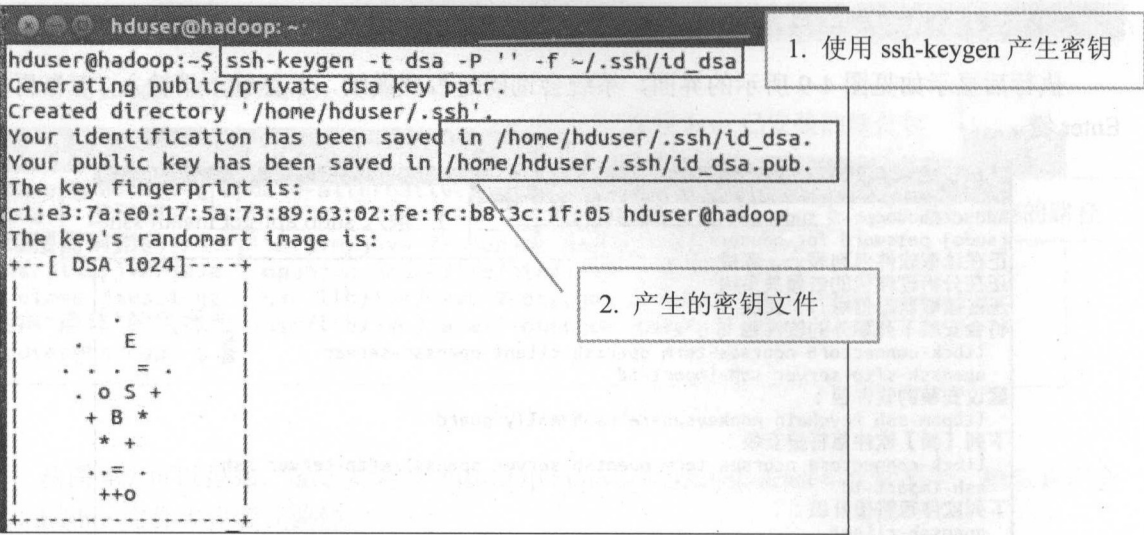


图 4-11 产生 SSH Key（密钥）

步骤 04 查看产生的 SSH Key（密钥）

SSH Key（密钥）会产生在用户的根目录下，也就是/home/hduser。

➤ 查看产生的 SSH Key（密钥）

```
ll ~/.ssh
```

运行后界面如图 4-12 所示。

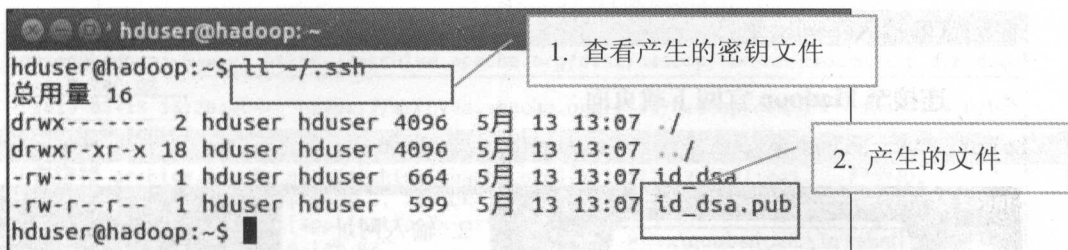


图 4-12 查看产生的 SSH Key (密钥)

步骤 05 将产生的 Key 放置到许可证文件中

为了能够无密码登录本机，我们必须将产生的公钥加入到许可证文件中。
在终端程序提示符输入以下命令：

➤ 将产生的 Key 放置到许可证文件中

```
cat ~/.ssh/id_dsa.pub >> ~/.ssh/authorized_keys
```

命令执行后，会将 ~/.ssh/id_dsa.pub 附加到 ~/.ssh/authorized_keys 许可证文件之后。

提示

Linux 的输出重定向附加功能命令的格式如下：

命令 >> 文件

这个重定向符号“>>”会将命令执行后产生的标准输出 (stdout) 重定向附加在该文件之后。

1. 如果文件不存在，就会先创建一个新文件，然后把标准输出 (stdout) 的内容存储在这个文件中。
2. 如果文件已经存在，就会将标准输出 (stdout) 的数据附加至文件内容的后面，而不会覆盖原来文件的内容。

运行后显示界面如图 4-13 所示。

将公钥放置到许可证文件中

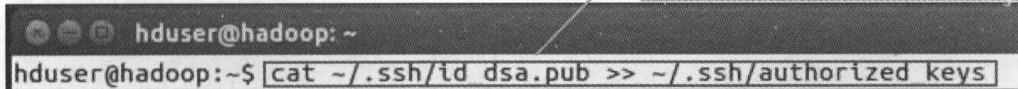


图 4-13 将产生的 Key 放置到许可证文件中

4.3 下载安装 Hadoop

接下来到 Hadoop 官网下载 Hadoop 64/32 bit 版本（根据操作系统版本来定），并且安装到 Ubuntu 中。

步骤 01 连接至 Hadoop 下载页面（见图 4-14）

在浏览器输入下列网址：

➤ 连接至 Hadoop 官网下载页面



图 4-14 连接至 Hadoop 下载页面

步骤 02 复制链接（见图 4-15）

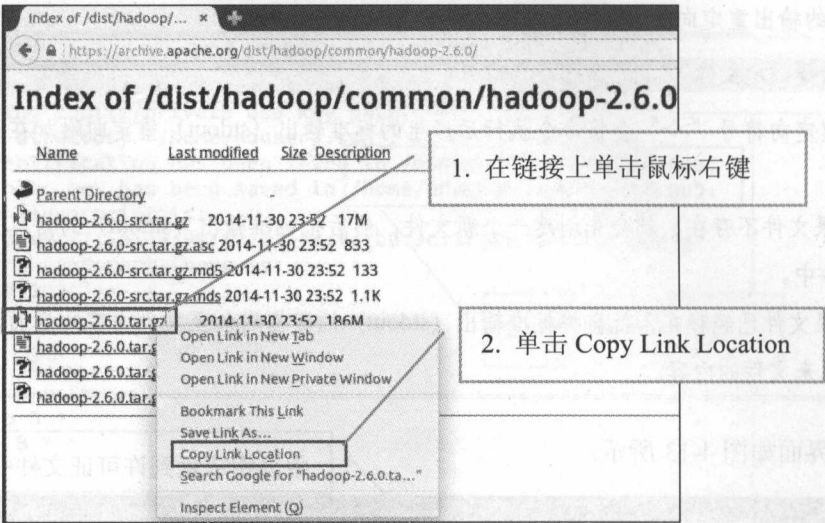


图 4-15 复制链接

步骤 03 执行下载命令

在“终端”程序提示符下输入 wget 及空格键，然后粘贴之前复制的链接，如下列命令：

➤ 下载 Hadoop-2.6.0.tar.gz

```
Wget https://archive.apache.org/dist/hadoop/common/hadoop-2.6.0/hadoop-2.6.0.tar.gz
```

执行结果如图 4-16 所示。



```

hduser@hadoop: ~
hduser@hadoop:~$ wget https://archive.apache.org/dist/hadoop/common/hadoop-2.6.0/hadoop-2.6.0.tar.gz
--2016-05-13 13:20:40-- https://archive.apache.org/dist/hadoop/common/hadoop-2.6.0/hadoop-2.6.0.tar.gz
正在解析主机 archive.apache.org (archive.apache.org)... 163.172.17.199
正在连接 archive.apache.org (archive.apache.org)|163.172.17.199|:443... 已连接。
已发出 HTTP 请求，正在等待回应... 200 OK
长度：195257604 (186M) [application/x-gzip]
正在保存至：“hadoop-2.6.0.tar.gz”

100%[=====] 195,257,604 3.40MB/s 用时 58s
2016-05-13 13:21:40 (3.20 MB/s) - 已保存 “hadoop-2.6.0.tar.gz” [195257604/195257604]

```

图 4-16 执行下载命令

步骤 04 解压缩 hadoop 2.6

在“终端”程序提示符输入以下命令：

➤ 解压缩 hadoop-2.6.0.tar.gz 至 hadoop-2.6.0 目录

```
sudo tar -zxvf Hadoop-2.6.0.tar.gz
```

执行结果如图 4-17 所示。

```

hduser@hadoop: ~
hduser@hadoop:~$ sudo tar -zxvf hadoop-2.6.0.tar.gz

```

图 4-17 解压缩 hadoop 2.6

步骤 05 将 Hadoop 移动到/usr/local

软件默认的安装路径就是在/usr/local，可在“终端”程序提示符下输入以下命令：

➤ 移动 hadoop-2.6.0 目录到/usr/local/hadoop

```
sudo mv hadoop-2.6.0 /usr/local/hadoop
```

执行后界面如图 4-18 所示。

```

hduser@hadoop: ~
hduser@hadoop:~$ sudo mv hadoop-2.6.0 /usr/local/hadoop

```

图 4-18 将 Hadoop 移动到/usr/local

步骤 06 查看 Hadoop 安装目录/usr/local/hadoop

在“终端”程序中输入下列命令：

➤ 查看 Hadoop 安装目录/usr/local/hadoop

```
ll /usr/local/hadoop
```

执行后界面如图 4-19 所示。

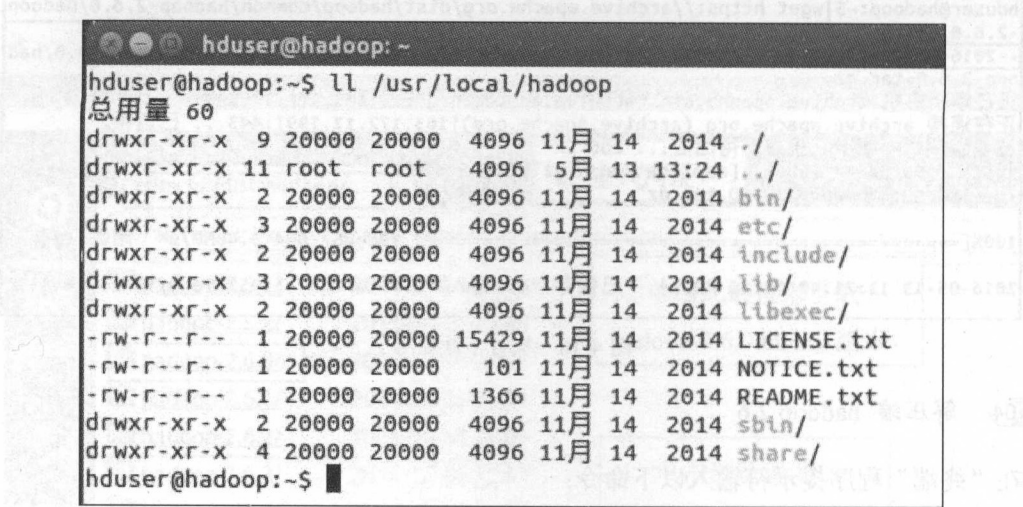


图 4-19 查看 Hadoop 安装目录/usr/local/hadoop

常用的目录说明如下：

目录	说明
bin/	各项运行文件，包括 Hadoop、HDFS、YARN 等
sbin/	各项 shell 运行文件，包括 start-all.sh、stop-all.sh
etc/	etc/hadoop 子目录包含 Hadoop 配置文件，例如 hadoop-env.sh、core-site.xml、yarn-site.xml、mapred-site.xml、hdfs-site.xml
lib/	Hadoop 函数库
logs/	系统日志，可以查看系统运行状况，运行有问题时可从日志找出错误原因

4.4 设置 Hadoop 环境变量

运行 Hadoop 必须设置很多环境变量，可是如果每次登录时都必须重新设置就会很麻烦。因此我们可以在 ~/.bashrc 文件中设置每次登录时都会自动运行一次环境变量设置。

步骤 01 编辑 ~/.bashrc

在“终端”程序中输入下列命令：

➤ 编辑 ~/.bashrc

```
sudo gedit ~/.bashrc
```

输入后按 Enter 键就会打开 ~/.bashrc，如图 4-20 所示。

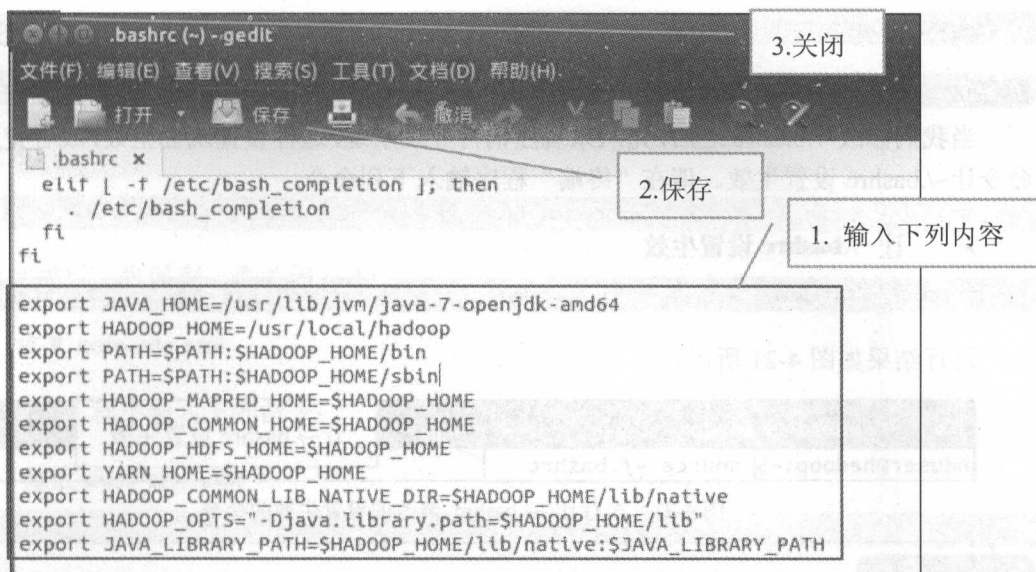


图 4-20 编辑 ~/.bashrc

编辑完成后，先保存，再关闭 gedit。

上列设置说明如下：

➤ 设置 JDK 安装路径（参考第 4.2 节）

```
export JAVA_HOME=/usr/lib/jvm/Java-7-openjdk-amd64
```

➤ 设置 HADOOP_HOME 为 Hadoop 的安装路径 /usr/local/hadoop

```
export HADOOP_HOME=/usr/local/hadoop
```

➤ 设置 PATH

```
export PATH=$PATH:$HADOOP_HOME/bin
export PATH=$PATH:$HADOOP_HOME/sbin
```

Hadoop 运行文件目录是 /bin 与 /sbin。设置 PATH 可以让你在其他目录时仍然能够运行 Hadoop 命令。

➤ 设置 HADOOP 其他环境变量

设置这些环境变量为 \$HADOOP_HOME，也就是 Hadoop 的安装路径 /usr/local/hadoop。

```
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export YARN_HOME=$HADOOP_HOME
```

➤ 链接库的相关设置

```
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib"
```



```
export JAVA_LIBRARY_PATH=$HADOOP_HOME/lib/native:$JAVA_LIBRARY_PATH
```

步骤 02 让 ~/.bashrc 修改的设置生效

当我们修改 ~/.bashrc 之后, 先从系统注销再登录系统, 这样设置就会生效, 或者使用 source 命令让 ~/.bashrc 设置生效。请在“终端”程序输入下列命令:

➤ 让 ~/.bashrc 设置生效

```
source ~/.bashrc
```

运行结果如图 4-21 所示。

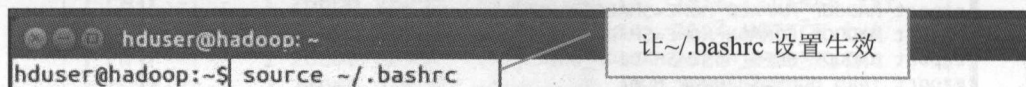


图 4-21 执行让 ~/.bashrc 修改的设置生效的命令

4.5 修改 Hadoop 配置设置文件

接下来要进行 Hadoop 配置设置, 包括 Hadoop-env.sh、core-site.xml、yarn-site.xml、mapred-site.xml、hdfs-site.xml。

步骤 01 设置 hadoop-env.sh 配置文件

hadoop-env.sh 是 Hadoop 的配置文件, 在这里必须设置 Java 的安装路径, 在“终端”程序输入下列命令:

➤ 编辑 Hadoop-env.sh

```
sudo gedit /usr/local/hadoop/etc/hadoop/hadoop-env.sh
```

输入后按 Enter 键就会打开 hadoop-env.sh, 屏幕显示界面如图 4-22 所示。

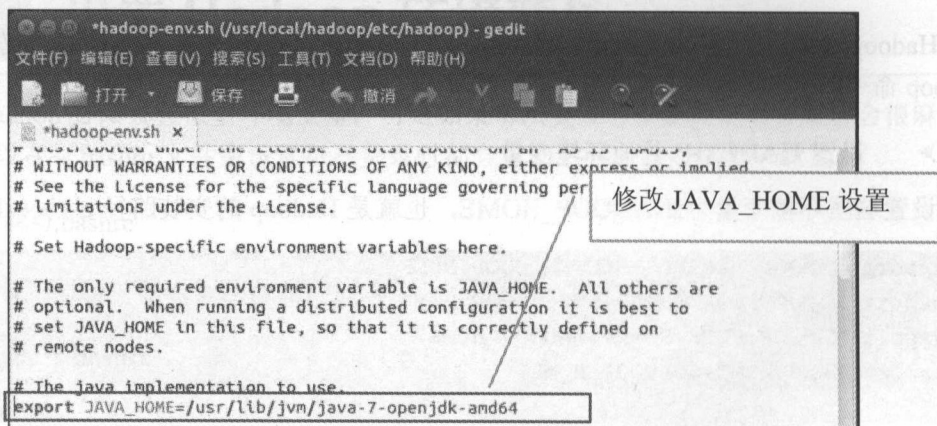


图 4-22 设置 hadoop-env.sh 配置文件

原本文件中 JAVA_HOME 的设置是：

```
export JAVA_HOME=${JAVA_HOME}
```

修改为：

```
export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64
```

编辑完成后，先保存，再关闭 gedit。

步骤 02 设置 core-site.xml

在“终端”程序输入下列命令：

➤ 修改 core-site.xml

```
sudo gedit /usr/local/hadoop/etc/hadoop/core-site.xml
```

输入后按 Enter 键就会打开 core-site.xml，屏幕显示界面如图 4-23 所示。

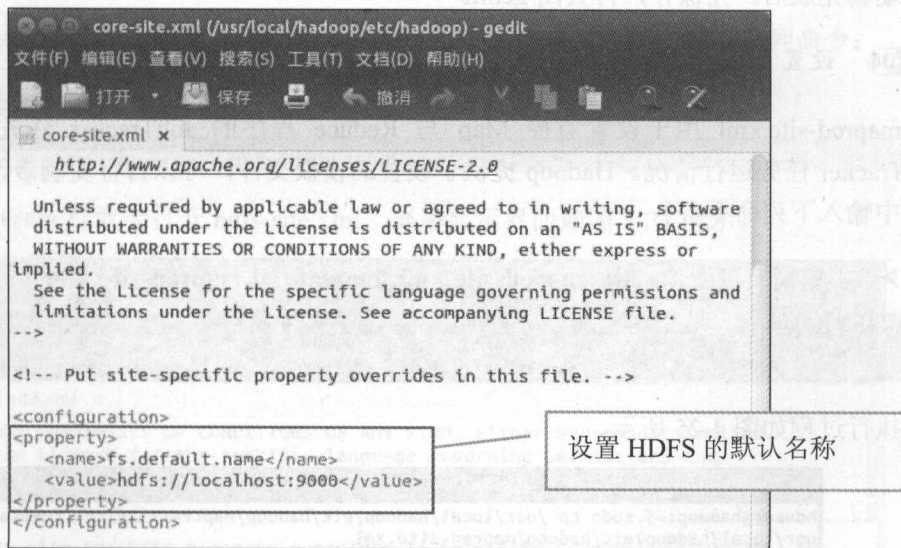


图 4-23 设置 core-site.xml

在 core-site.xml 中，我们必须设置 HDFS 的默认名称，当我们使用命令或程序要存取 HDFS 时，可使用此名称。编辑完成后，先保存，再关闭 gedit。

步骤 03 设置 yarn-site.xml

yarn-site.xml 文件中含有 MapReduce2 (YARN) 相关的配置设置。可在“终端”程序输入下列命令：

➤ 编辑 yarn-site.xml

```
sudo gedit /usr/local/hadoop/etc/hadoop/yarn-site.xml
```

输入后按 Enter 键就会打开 yarn-site.xml，屏幕显示界面如图 4-24 所示。

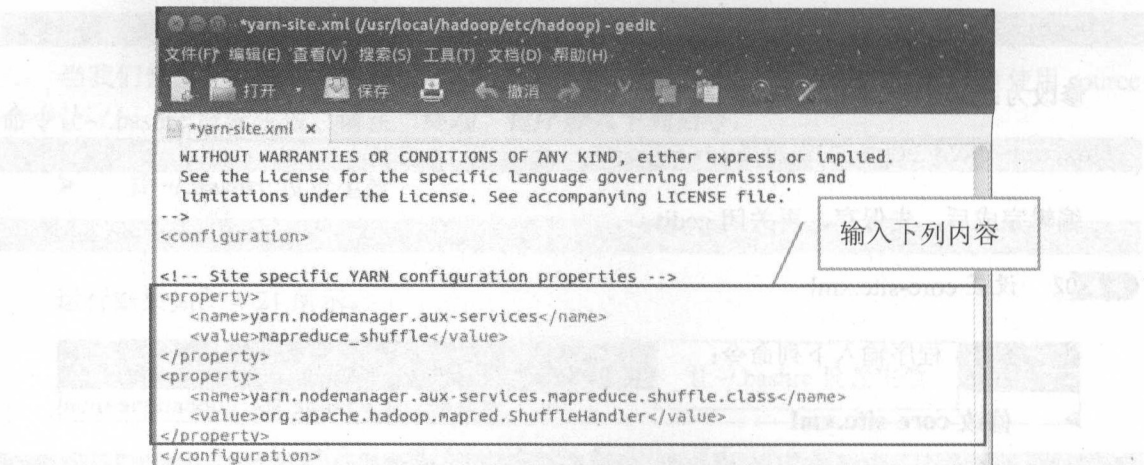


图 4-24 设置 yarn-site.xml

编辑完成后，先保存，再关闭 gedit。

步骤 04 设置 mapred-site.xml

mapred-site.xml 用于设置监控 Map 与 Reduce 程序的 JobTracker 任务分配情况以及 TaskTracker 任务运行情况。Hadoop 提供了设置的模板文件，可以自行复制修改。在“终端”程序中输入下列命令：

➤ 复制模板文件：由 `mapred-site.xml.template` 至 `mapred-site.xml`

```
sudo cp /usr/local/hadoop/etc/hadoop/mapred-site.xml.template
/usr/local/hadoop/etc/hadoop/mapred-site.xml
```

执行过程如图 4-25 所示。

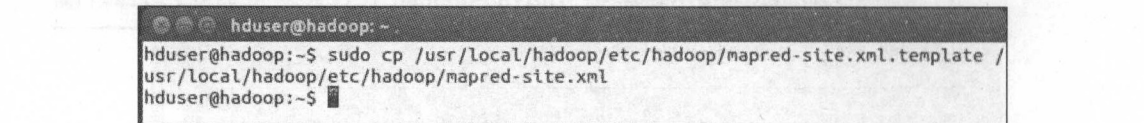


图 4-25 设置 mapred-site.xml

➤ 编辑 mapred-site.xml

```
sudo gedit /usr/local/hadoop/etc/hadoop/mapred-site.xml
```

输入后按 Enter 键就会打开 mapred-site.xml。屏幕显示界面如图 4-26 所示。

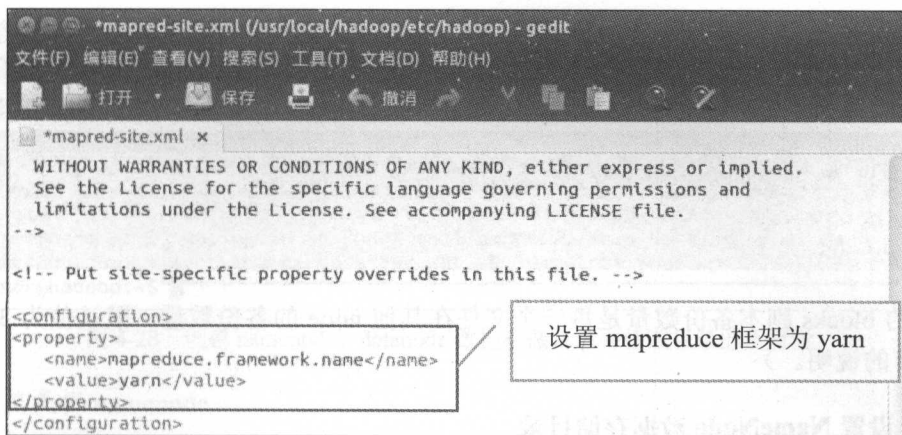


图 4-26 打开 mapred-site.xml 进行编辑

编辑完成后，先保存，再关闭 gedit。

步骤 05 设置 hdfs-site.xml

hdfs-site.xml 用于设置 HDFS 分布式文件系统，在“终端”程序中输入下列命令：

➤ 编辑 hdfs-site.xml

```
sudo gedit /usr/local/hadoop/etc/hadoop/hdfs-site.xml
```

输入后按 Enter 键就会打开 hdfs-site.xml，屏幕显示界面如图 4-27 所示。

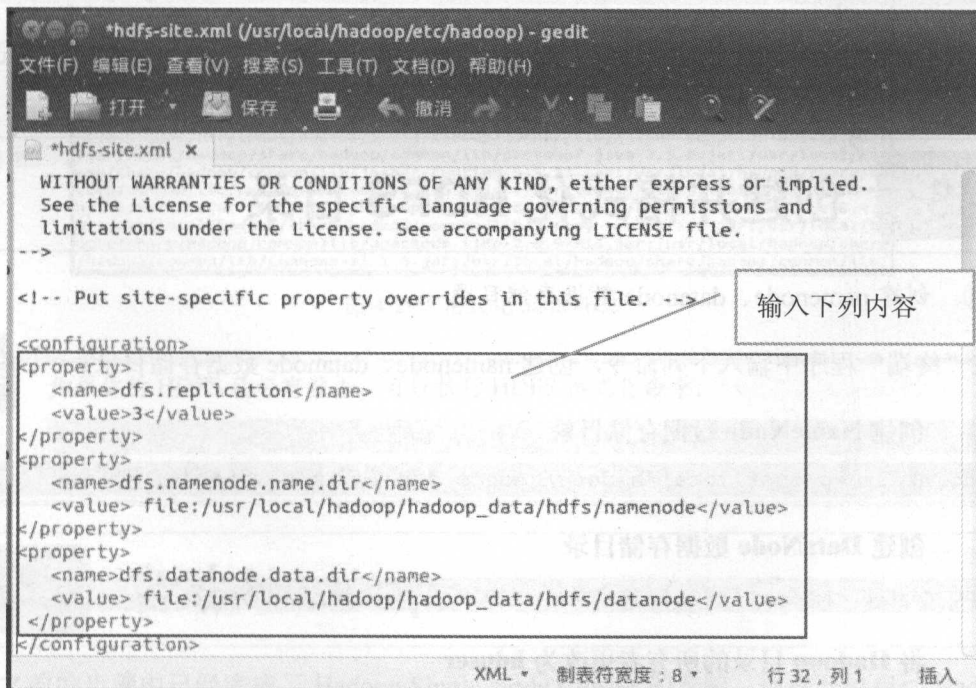


图 4-27 设置 hdfs-site.xml

打开 `hdfs-site.xml` 后，在 `<configuration>` `</configuration>` 标签之间输入下列内容：

➤ 设置 **blocks** 副本备份数量

```
<property>
  <name>dfs.replication</name>
  <value>3</value>
</property>
```

默认的 **blocks** 副本备份数量是每一个文件在其他 **node** 的备份数量，默认值为 3。（可参考第 1.3 节的说明。）

➤ 设置 **NameNode** 数据存储目录

```
<property>
  <name>dfs.namenode.name.dir</name>
  <value> file:/usr/local/hadoop/hadoop_data/hdfs/namenode</value>
</property>
```

设置 **NameNode** 数据存储目录为 `/usr/local/hadoop/hadoop_data/hdfs/namenode`。

➤ 设置 **DataNode** 数据存储

```
<property>
  <name>dfs.datanode.data.dir</name>
  <value> file:/usr/local/hadoop/hadoop_data/hdfs/datanode</value>
</property>
```

设置 **DataNode** 数据存储目录为 `/usr/local/hadoop/hadoop_data/hdfs/datanode`。编辑完成后，先保存，再关闭 `gedit`。

4.6 创建并格式化 HDFS 目录

步骤 01 创建 **namenode**、**datanode** 数据存储目录

在“终端”程序中输入下列命令，创建 **namenode**、**datanode** 数据存储目录：

➤ 创建 **NameNode** 数据存储目录

```
sudo mkdir -p /usr/local/hadoop/hadoop_data/hdfs/namenode
```

➤ 创建 **DataNode** 数据存储目录

```
sudo mkdir -p /usr/local/hadoop/hadoop_data/hdfs/datanode
```

➤ 将 **Hadoop** 目录的所有者更改为 **hduser**

```
sudo chown hduser:hduser -R /usr/local/hadoop
```


Linux 是多人多任务的操作系统，所有的目录或文件都具有所有者。使用 `chown` 可以将目录或文件的所有者更改为 `hduser`。

执行结果界面如图 4-28 所示。

```
hduser@hadoop: ~
hduser@hadoop:~$ sudo mkdir -p /usr/local/hadoop/hadoop_data/hdfs/namenode
hduser@hadoop:~$ sudo mkdir -p /usr/local/hadoop/hadoop_data/hdfs/datanode
hduser@hadoop:~$ sudo chown hduser:hduser -R /usr/local/hadoop
hduser@hadoop:~$
```

图 4-28 创建 namenode、datanode 数据存储目录并修改目录的所有者

步骤 02 格式化 namenode

在“终端”程序中输入下列命令：

➤ 将 HDFS 进行格式化

```
hadoop namenode -format
```

执行结果界面如图 4-29 所示。

```
hduser@hadoop: ~
hduser@hadoop:~$ hadoop namenode -format
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.

15/04/28 20:58:22 INFO namenode.NameNode: STARTUP_MSG:
/*****
STARTUP_MSG: Starting NameNode
STARTUP_MSG: host = hadoop/127.0.1.1
STARTUP_MSG: args = [-format]
STARTUP_MSG: version = 2.6.0
STARTUP_MSG: classpath = /usr/local/hadoop/etc/hadoop:/usr/local/hadoop/share/
hadoop/common/lib/log4j-1.2.17.jar:/usr/local/hadoop/share/hadoop/common/lib/jac
kson-xc-1.9.13.jar:/usr/local/hadoop/share/hadoop/common/lib/paranamer-2.3.jar:/
usr/local/hadoop/share/hadoop/common/lib/zookeeper-3.4.6.jar:/usr/local/hadoop/s
hare/hadoop/common/lib/jackson-core-asl-1.9.13.jar:/usr/local/hadoop/share/hadoo
p/common/lib/jettison-1.1.jar:/usr/local/hadoop/share/hadoop/common/lib/httpcore
-4.2.5.jar:/usr/local/hadoop/share/hadoop/common/lib/jasper-compiler-5.5.23.jar:
/usr/local/hadoop/share/hadoop/common/lib/protobuf-java-2.5.0.jar:/usr/local/had
oop/share/hadoop/common/lib/jasper-runtime-5.5.23.jar:/usr/local/hadoop/share/ha
doo/common/lib/curator-framework-2.6.0.jar:/usr/local/hadoop/share/hadoop/commo
n/lib/xz-1.0.jar:/usr/local/hadoop/share/hadoop/common/lib/java-xmlbuilder-0.4.j
ar:/usr/local/hadoop/share/hadoop/common/lib/jersey-server-1.9.jar:/usr/local/ha
doo/share/hadoop/common/lib/apacheds-l18n-2.0.0-M15.jar:/usr/local/hadoop/share
/hadoop/common/lib/commons-el-1.0.jar:/usr/local/hadoop/share/hadoop/common/lib/
```

图 4-29 格式化 namenode

提示

如果你的 HDFS 已经有数据，可以执行 HDFS 格式化命令：

```
hadoop namenode -format
```

这个操作会删除所有的数据。

4.7 启动 Hadoop

在之前的步骤中已经完成了 Hadoop Single Node Cluster 的安装，现在开始启动 Hadoop。可以使用以下两种方式：

- 分别启动 HDFS、YARN，使用 start-dfs.sh（启动 HDFS）、start-yarn.sh（启动 YARN）
- 同时启动 HDFS、YARN，使用 start-all.sh。

步骤 01 启动 HDFS

在“终端”程序中输入下列命令：

➤ 启动 HDFS

```
start-dfs.sh
```

执行结果的屏幕显示界面如图 4-30 所示。

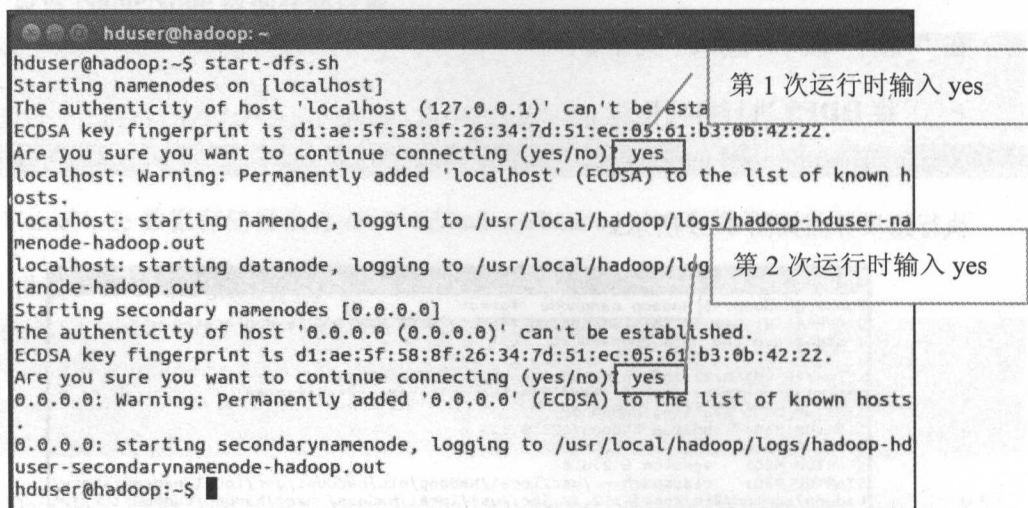


图 4-30 启动 HDFS

步骤 02 启动 YARN

在“终端”程序中输入下列命令：

➤ 启动 Hadoop MapReduce 框架 YARN

```
start-yarn.sh
```

执行结果的屏幕显示界面如图 4-31 所示。

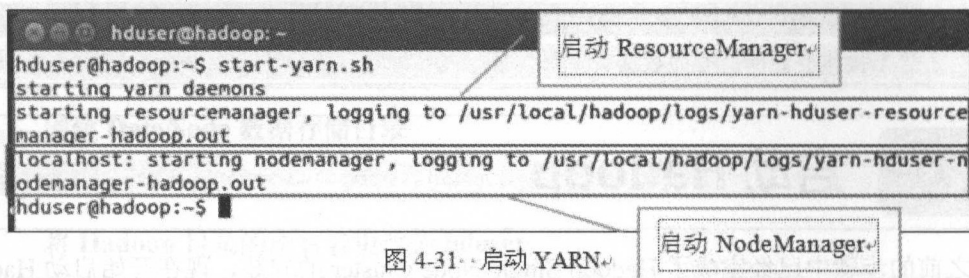


图 4-31 启动 YARN

步骤 03 同时启动 HDFS 和 YARN: start-all.sh

另外, 还可以使用 start-all.sh。

➤ 同时启动 HDFS、YARN

```
start-all.sh
```

执行结果的屏幕显示界面如图 4-32 所示。

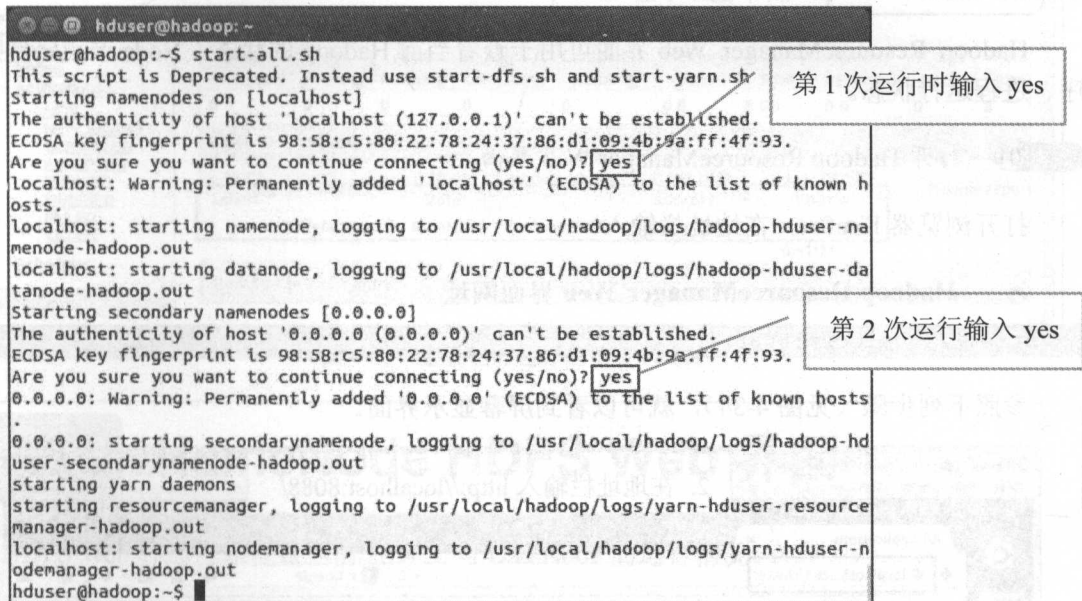


图 4-32 同时启动 HDFS 和 YARN: start-all.sh

步骤 04 使用 jps 查看已经启动的进程

jps (Java Virtual Machine Process Status Tool), 可以查看当前所运行的进程 (process), 在“终端”程序中输入下列命令:

➤ 查看 NameNode、DataNode 进程是否启动

```
jps
```

执行后屏幕显示界面如图 4-33 所示。

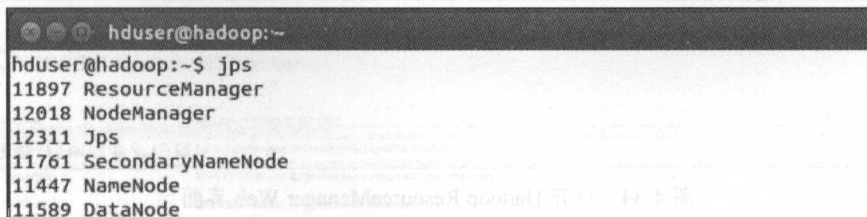


图 4-33 使用 jps 查看已经启动的进程

因为只有一台服务器，所以所有功能都集中在一台服务器中，你可以看到：

- **HDFS 功能：** NameNode、Secondary NameNode、DataNode 已经启动。
- **MapReduce2 (YARN)：** Resource Manager、NodeManager 已经启动。

4.8 打开 Hadoop Resource-Manager Web 界面

Hadoop ResourceManager Web 界面可用于查看当前 Hadoop 的状态：Node 节点、应用程序、进程运行状态。

步骤 01 打开 Hadoop ResourceManager Web 界面

打开浏览器 Firefox，在地址栏输入：

➤ Hadoop ResourceManager Web 界面网址

`http://localhost:8088/`

参照下列步骤（见图 4-34），就可以看到屏幕显示界面。

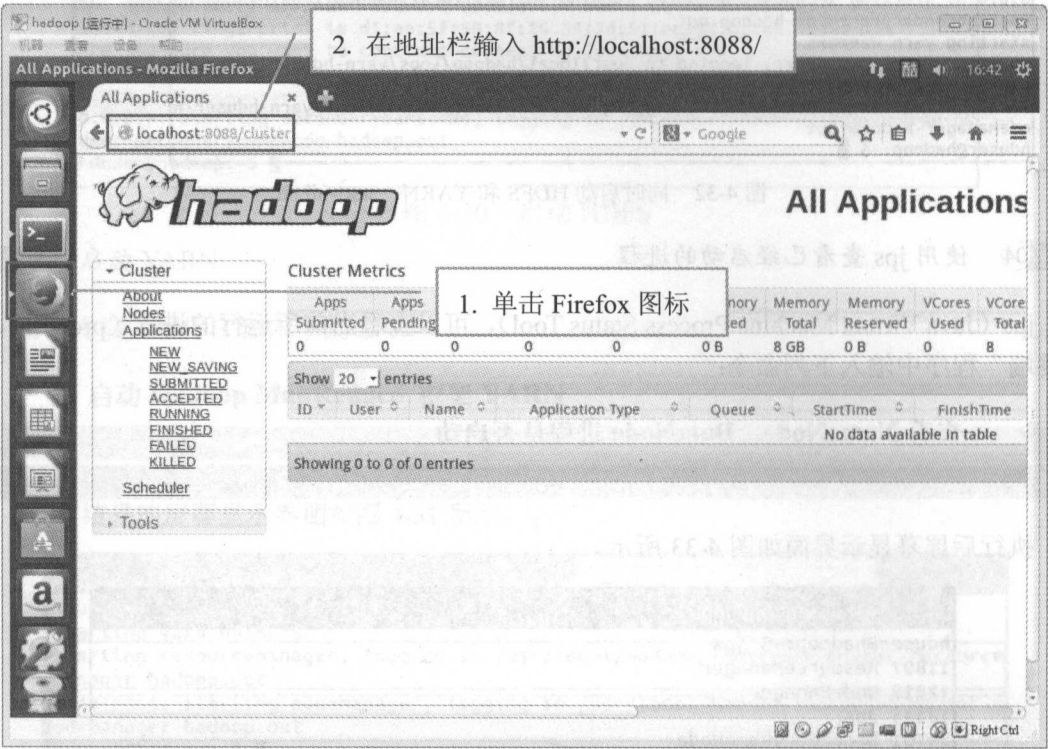


图 4-34 打开 Hadoop ResourceManager Web 界面

步骤 02 查看已经运行的节点 Nodes

当你单击 Nodes 时，会显示当前的节点。不过，因为我们安装的是 Single Node Cluster，所以当前只有一个节点，如图 4-35 所示。

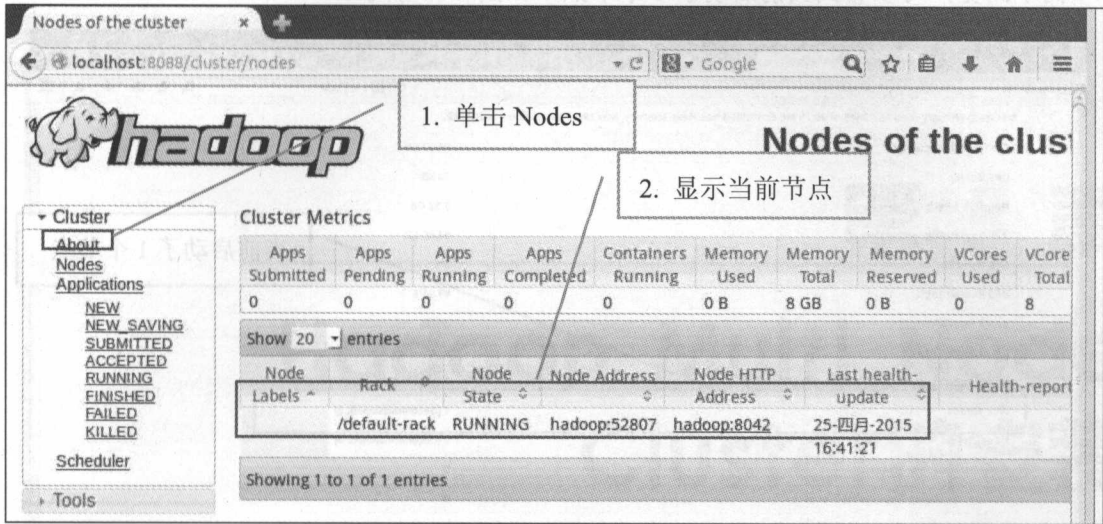


图 4-35 查看已经运行的节点 Nodes

4.9 NameNode HDFS Web 界面

HDFS Web 界面可以检查当前 HDFS 与 DataNode 的运行情况。

步骤 01 打开 NameNode HDFS Web 界面

打开浏览器 Firefox，在地址栏输入：

➤ **HDFS Web UI 网址**

http://localhost:50070/

可以看到屏幕显示界面如图 4-36 所示。

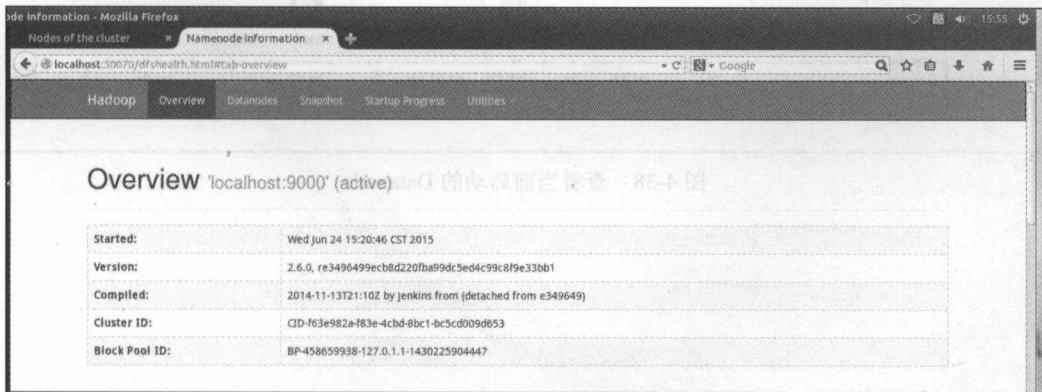


图 4-36 打开 NameNode HDFS Web 界面

步骤 02 查看 Live Nodes

向下浏览，可以看到当前启动了 1 个节点，如图 4-37 所示。

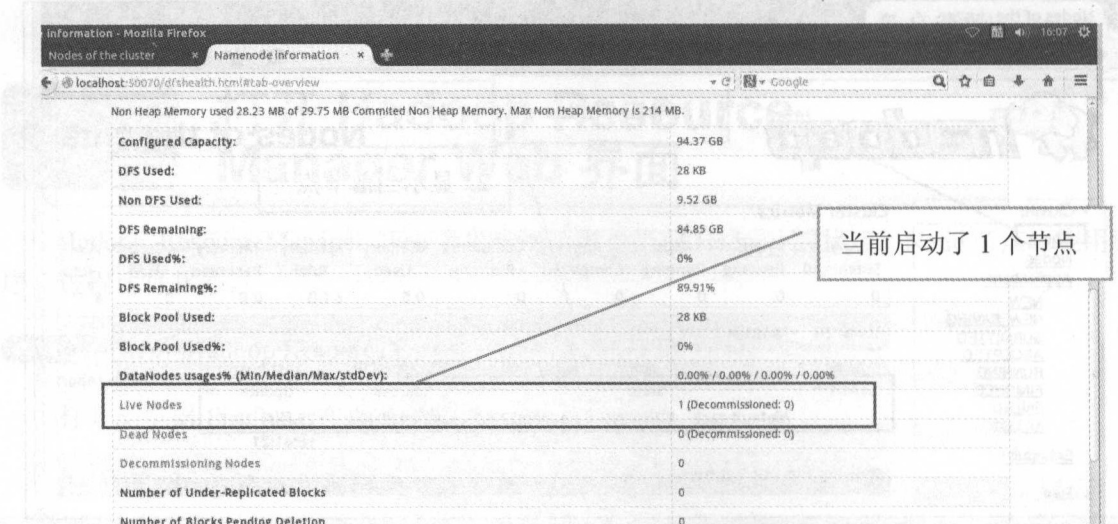


图 4-37 查看 Live Nodes（启动的节点数）

步骤 03 查看 DataNodes（见图 4-38）

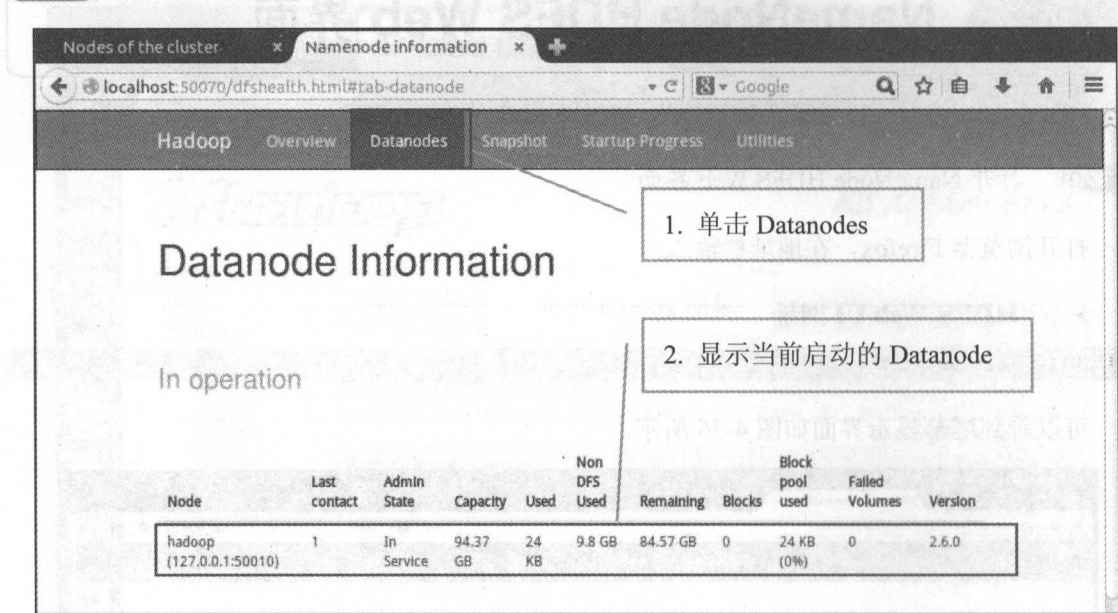


图 4-38 查看当前启动的 Datanode

第 5 章

Hadoop Multi Node Cluster的安装

- 5.1 把 Single Node Cluster 复制到 data1
- 5.2 设置 VirtualBox 网卡
- 5.3 设置 data1 服务器
- 5.4 复制 data1 服务器到 data2、data3、master
- 5.5 设置 data2、data3 服务器
- 5.6 设置 master 服务器
- 5.7 master 连接到 data1、data2、data3 创建 HDFS 目录
- 5.8 创建并格式化 NameNode HDFS 目录
- 5.9 启动 Hadoop Multi Node Cluster
- 5.10 打开 Hadoop ResourceManager Web 界面
- 5.11 打开 NameNode Web 界面

Hadoop Multi Node Cluster 规划如图 5-1 所示，由多台计算机组成：

- 有一台主要的计算机 master，在 HDFS 担任 NameNode 角色、在 MapReduce2(YARN) 担任 ResourceManager 角色。
- 有多台的计算机 data1、data2、data3，在 HDFS 担任 DataNode 角色、在 MapReduce2(YARN) 担任 NodeManager 角色。

Hadoop Multi Node Cluster 规划，整理如下表格所示：

服务器名称	内部 IP	HDFS	YARN
master	192.168.56.100	NameNode	ResourceManager
data1	192.168.56.101	DataNode	NodeManager
data2	192.168.56.102	DataNode	NodeManager
data3	192.168.56.103	DataNode	NodeManager

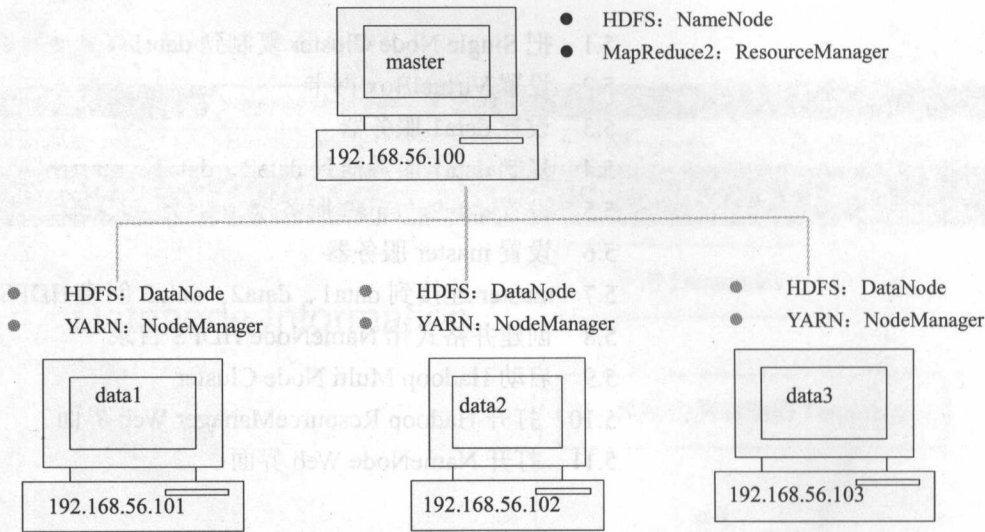


图 5-1 Hadoop Multi Node Cluster 架构的示意图

图 5-1 的 Hadoop Multi Node Cluster 架构，必须有 4 台实体服务器才能建立，才能够发挥多台计算机并行处理的优势。

但是考虑大部分的读者，只有一台个人计算机，为了方便大家实机演练，所以我们使用 VirtualBox 创建 4 台虚拟主机 master、data1、data2、data3。因为是虚拟主机，所以你无法享受到多台计算机并行处理的优势。不过，你在虚拟主机所学到的设置，完全可以用于创建实体主机 Hadoop Multi Node Cluster。

Hadoop Multi Node Cluster 的安装步骤如下：

顺序	安装步骤	说明
1	复制 Single Node Cluster 到 data1	为了节省安装时间，我们将上一章所创建的 Single Node Cluster Hadoop 复制到 data1，创建 data1 虚拟机
2	设置 data1 服务器	设置 Multi Node Cluster 服务器；配置设置文件共同的部分：固定 ip、hostname、core-site.xml、yarn-site.xml、mapred-site.xml、hdfs-site.xml
3	复制 data1 服务器至 data2、data3、master	我们已经设置了 data1 服务器，配置设置文件共同的部分，为了节省安装时间，所以复制 data1 到 data2、data3、master
4	设置 data2、data3 服务器	设置 data2、data3 固定 ip、hostname
5	设置 master 服务器	设置 NameNode 服务器（master）： 设置固定 ip、hostname、hdfs-site.xml、masters、slaves
6	master 连接到 data1、data2、data3 创建 HDFS 目录	重新启动 master 与 data1、data2、data3 后。master 通过 SSH 连接 data1、data2、data3 并创建 HDFS 目录
7	建立与格式化 NameNode HDFS 目录	创建 NameNode HDFS 目录，并且格式化 HDFS 目录
8	启动 Hadoop Multi Node cluster	启动 Hadoop Multi Cluster 并使用 jps 查看当前所运行的进程
9	打开 Hadoop ResourceManager Web 界面	Hadoop ResourceManagerWeb 界面可用于查看当前 Hadoop: Node 节点、应用程序、进程运行状态
10	打开 NameNode Web 界面	HDFS Web 界面，可以检查当前 HDFS 与 DataNode 运行情况

➤ Hadoop 2.6 Multi Node Cluster 安装命令

以下安装过程中所需要输入的命令，我们已整理在本书有关的博客文章中。当练习安装时，可以复制博客文章中的命令，然后粘贴到“终端”程序中。这样不但可以节省打字的时间，也不用担心打错字（如果无法在 VirtualBox 虚拟机的 Ubuntu“终端”程序进行复制/粘贴操作时，请参考第 3.9 节的说明，设置好 VirtualBox 的共享剪贴板）。本书的博客网址为：

<http://blog.sina.com.cn/hadoopsparkbook>

5.1 把 Single Node Cluster 复制到 data1

为了节省安装时间，我们将复制上一章所创建的 Single Node Cluster 来创建 data1 虚拟机。如果不使用复制虚拟机的方法，也可以重复上一章的步骤来创建 data1。

步骤 01 复制 Hadoop 到 data1（见图 5-2）

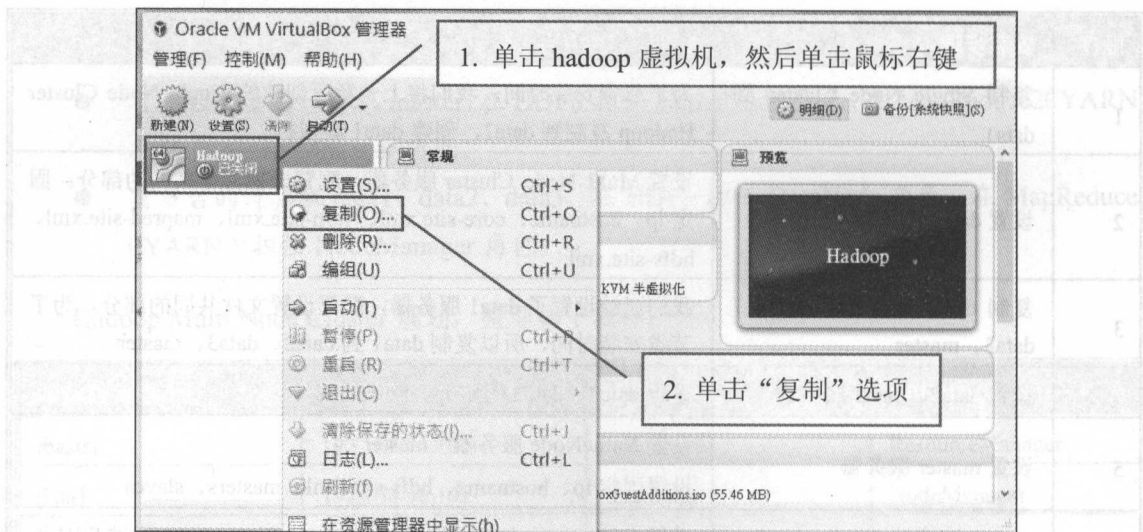


图 5-2 复制 Hadoop 到 data1

步骤 02 设置虚拟机名称（见图 5-3）

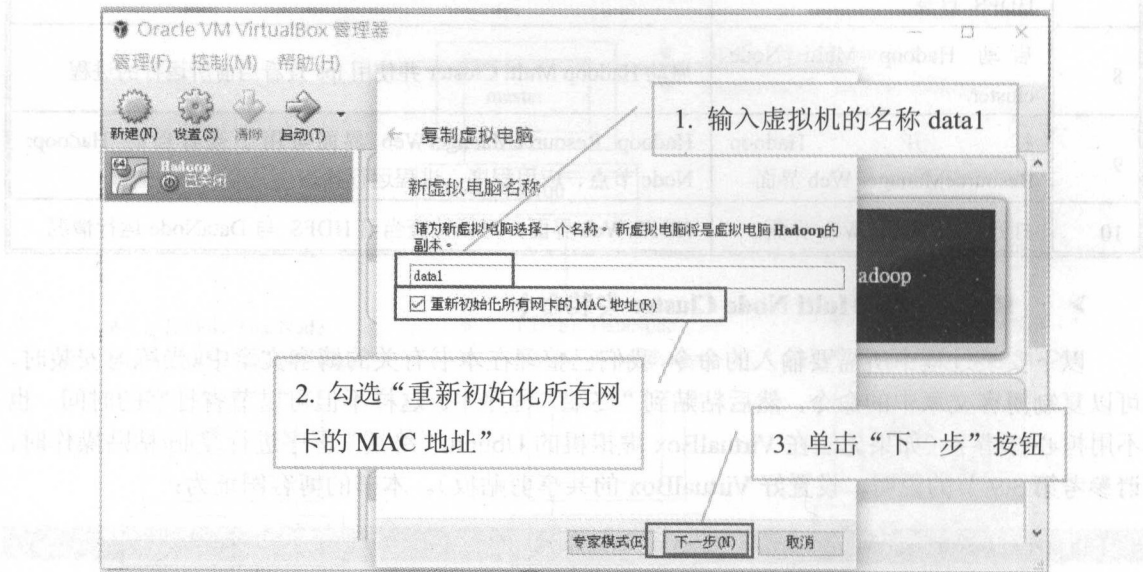


图 5-3 设置虚拟机的名称

步骤 03 设置复制类型

参照如图 5-4 所示的步骤，选择复制类型，再单击“复制”按钮后，大约需要数分钟等待。

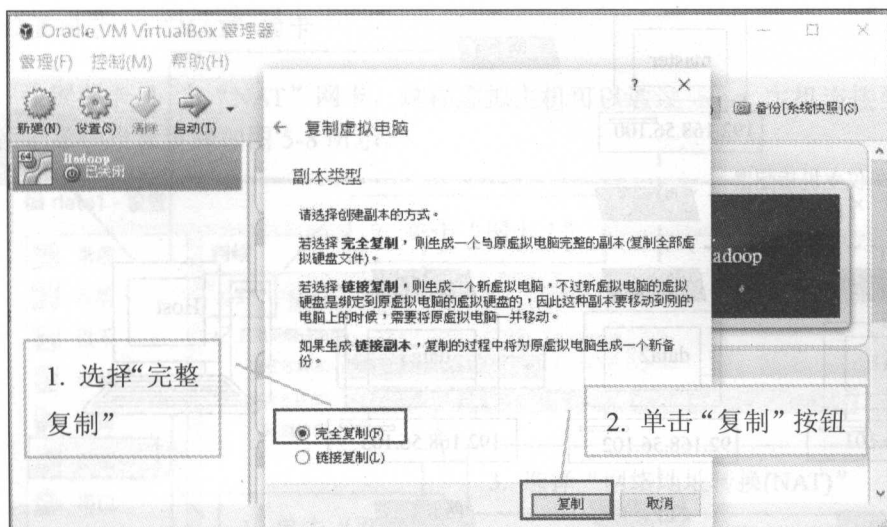


图 5-4 选择“复制”类型后开始复制

步骤 04 复制完成

复制完成后就会出现 data1，如图 5-5 所示。

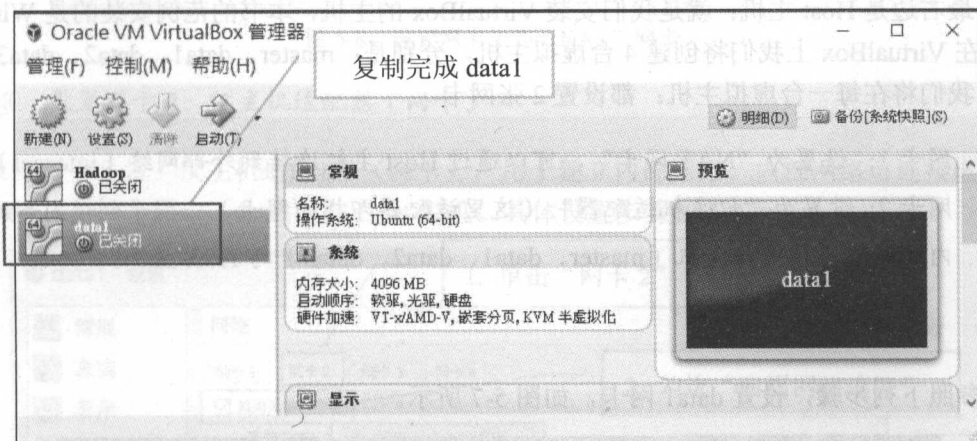


图 5-5 复制完成

5.2 设置 VirtualBox 网卡

我们将设置 VirtualBox 网卡，如图 5-6 所示。

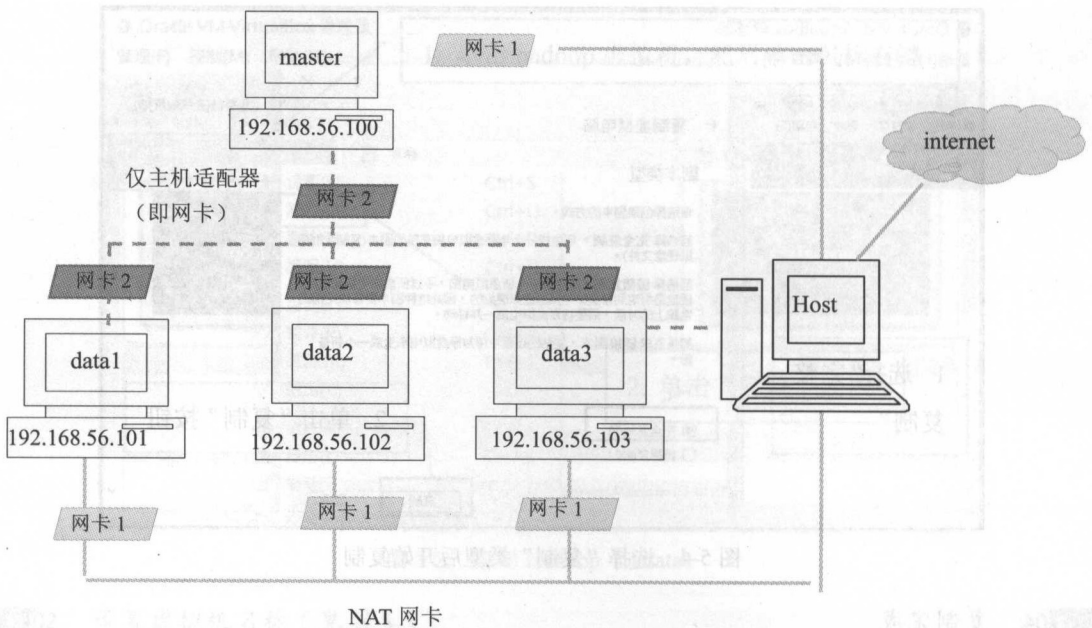


图 5-6 VirtualBox 网卡设置示意图

1. 最右边是 Host 主机：就是我们安装 VirtualBox 的主机，本书的范例安装的是 Windows 系统，在 VirtualBox 上我们将创建 4 台虚拟主机，分别是：master、data1、data2、data3。
2. 我们将在每一台虚拟主机：都设置 2 张网卡。
 - 网卡 1：设置为“NAT 网卡”，可以通过 Host 主机连接到外部网络（internet）。
 - 网卡 2：设置为“仅主机适配器”（这里适配器即指的网卡），用于创建内部网络，内部网络连接虚拟主机（master、data1、data2、data3）与 Host 主机。

步骤 01 设置网卡

请参照下列步骤，设置 data1 网卡，如图 5-7 所示。

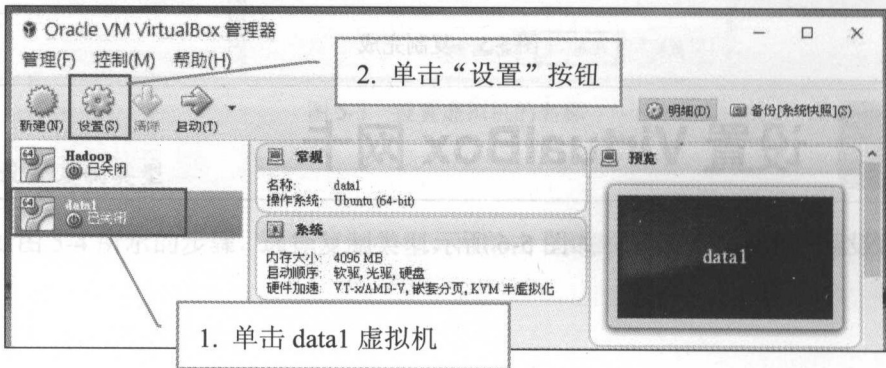


图 5-7 选择虚拟机，再选择“设置”按钮

步骤 02 设置网卡 1：“NAT”网卡

首先，设置网卡 1 为“NAT”网卡，这样虚拟主机可以通过 Host 主机连接至外部网络（internet）。具体设置步骤如图 5-8 所示。

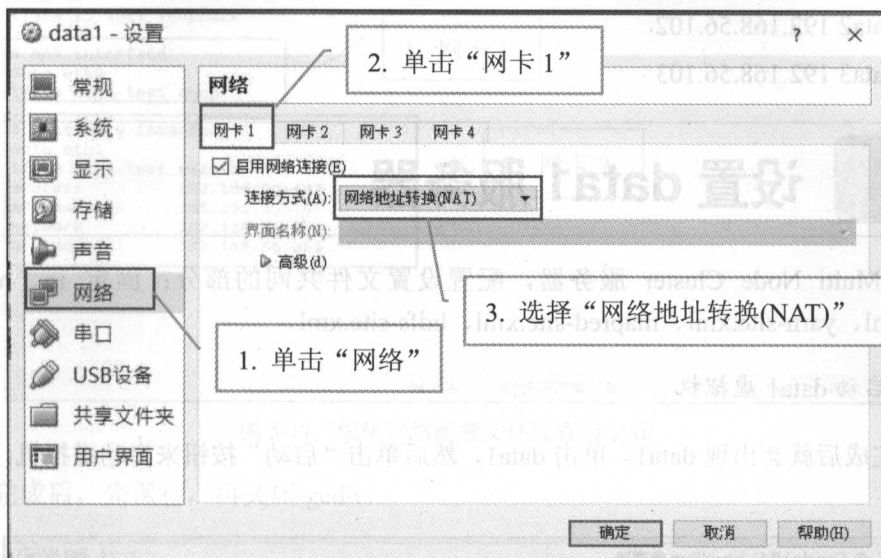


图 5-8 设置网卡 1 为“NAT”网卡

步骤 03 设置网卡 2：仅主机适配器（网卡）

设置网卡 2 为“仅主机适配器”（网卡），用于建立内部网络，内部网络可连接虚拟主机（master、data1、data2、data3）与 Host 主机。具体设置步骤如 5-9 所示。

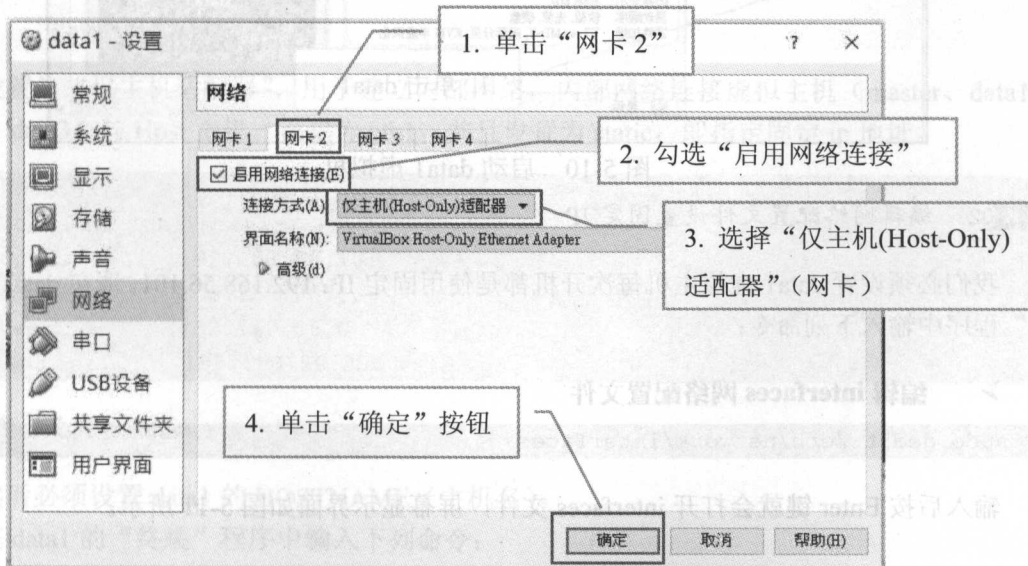


图 5-9 设置网卡 2 为“仅主机适配器”（网卡）

VirtualBox host-only 网卡 IP 段默认为 192.168.56.X，所以我们将设置虚拟主机的 IP 为

192.168.56.X，具体如下：

- master 192.168.56.100
- data1 192.168.56.101
- data2 192.168.56.102
- data3 192.168.56.103

5.3 设置 data1 服务器

设置 Multi Node Cluster 服务器，配置设置文件共同的部分：固定 ip、hostname、core-site.xml、yarn-site.xml、mapred-site.xml、hdfs-site.xml。

步骤 01 启动 data1 虚拟机

复制完成后就会出现 data1。单击 data1，然后单击“启动”按钮来启动虚拟机，如图 5-10 所示。

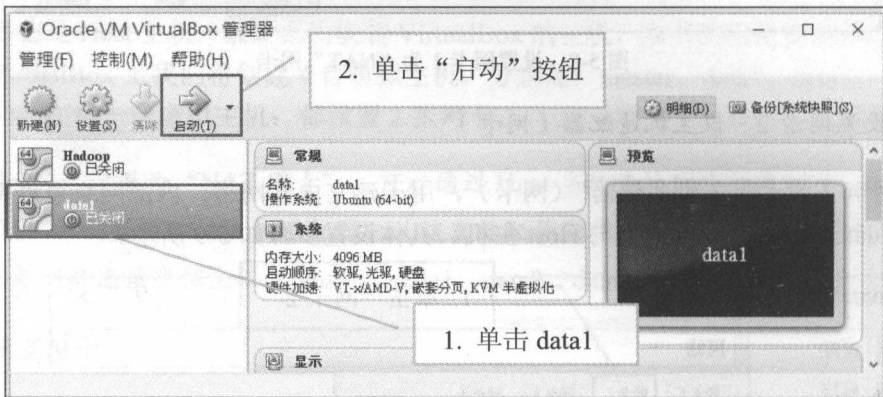


图 5-10 启动 data1 虚拟机

步骤 02 编辑网络配置文件设置固定 IP

我们必须设置 data1 虚拟主机每次开机都是使用固定 IP: 192.168.56.101。请在 data1 的“终端”程序中输入下列命令：

➤ 编辑 interfaces 网络配置文件

```
sudo gedit /etc/network/interfaces
```

输入后按 Enter 键就会打开 interfaces 文件，屏幕显示界面如图 5-11 所示。

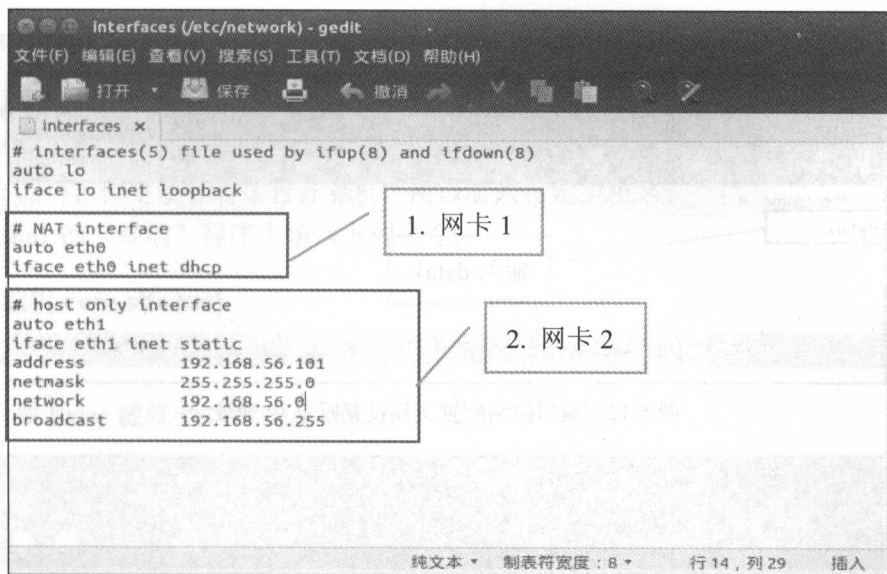


图 5-11 编辑网络配置文件设置固定 IP

编辑完成后，先保存，再关闭 gedit。

➤ 设置网卡 1

设置为“NAT 网卡”，可以通过 Host 主机连接至外部网络（internet），设置为 eth0，并且设置 dhcp 自动获得 ip 地址。

```
auto eth0
iface eth0 inet dhcp
```

➤ 设置网卡 2

设置为“仅主机适配器”，用于建立内部网络，内部网络连接虚拟主机（master、data1、data2、data3）与 Host 主机。设置为 eth1，并且设置为 static，即指定固定 ip 地址。

```
auto eth1
iface eth1 inet static
address      192.168.56.101
netmask      255.255.255.0
network      192.168.56.0
broadcast    192.168.56.255
```

步骤 03 设置 hostname

然后必须设置 data1 的 HOSTNAME（主机名）。

在 data1 的“终端”程序中输入下列命令：

➤ 编辑 hostname 主机名

```
sudo gedit /etc/hostname
```

输入后按 Enter 键就会打开 hostname，如图 5-12 所示。



图 5-12 编辑网络配置文件设置固定 IP 地址

编辑完成后，先保存，再关闭 gedit。

步骤 04 设置 hosts 文件

我们建立的 Hadoop Multi Node Cluster 中有 4 台计算机，如何让网络中所有的计算机，都知道其他计算机的主机名与 IP 呢？可以编辑 hosts 文件或设置 DNS。hosts 文件通常用于补充或取代网络中 DNS 的功能。和 DNS 不同的是，计算机的用户可以直接对 hosts 文件进行控制。hosts 文件可存储计算机网络中各节点的信息，负责将主机名映射到对应的 IP 地址（即名字解析）。

请在 data1 “终端” 程序中输入下列命令：

➤ 编辑 hosts 文件

```
sudo gedit /etc/hosts
```

输入后按 Enter 键就会打开 hosts 文件，请设置各节点的主机名与相对应的 IP 地址，如图 5-13 所示。

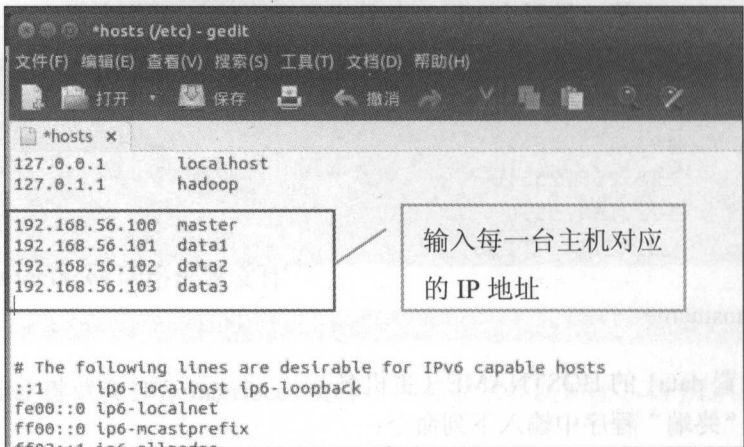


图 5-13 设置主机名和 IP 地址对应

编辑完成后，先保存，再关闭 gedit。

步骤 05 编辑 core-site.xml

在 core-site.xml 中,我们必须设置 HDFS 的默认名称,当使用命令或程序来存取 HDFS 时,可使用此名称。之前 Single Node Cluster 因为只有一台计算机,所以我们设置 namenode 位置为 localhost 即可,但是现在有多台计算机,所以必须指定主机名。

请在 data1 的“终端”程序中输入下列命令:

➤ 编辑 core-site.xml

```
sudo gedit /usr/local/hadoop/etc/hadoop/core-site.xml
```

输入后按 Enter 键就会打开 core-site.xml, 如图 5-14 所示。

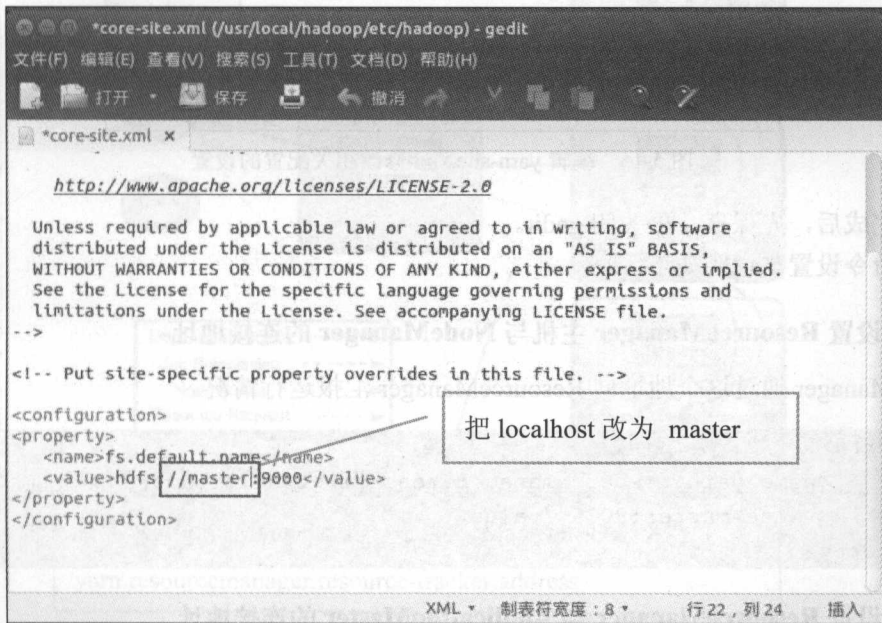


图 5-14 编辑 core-site.xml 指定主机名

编辑完成后,先保存,再关闭 gedit。之后当使用程序存取 HDFS 时,会使用 hdfs://master:9000 这个目标来存取 HDFS。

步骤 06 编辑 yarn-site.xml

yarn-site.xml 文件是 MapReduce2 (YARN) 相关配置的设置。

在 data1 的“终端”程序中输入下列命令:

➤ 编辑 yarn-site.xml

```
sudo gedit /usr/local/hadoop/etc/hadoop/yarn-site.xml
```

输入后按 Enter 键就会打开 yarn-site.xml, 屏幕显示界面如图 5-15 所示:

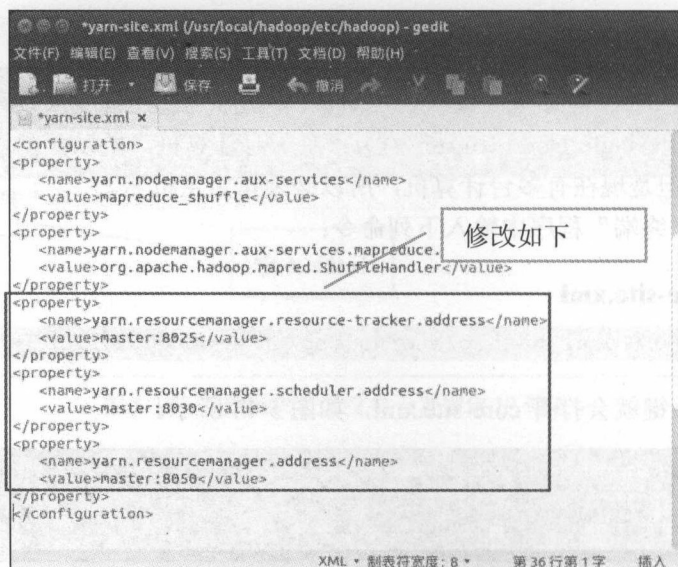


图 5-15 编辑 yarn-site.xml 修改相关配置的设置

编辑完成后，先保存，再关闭 gedit。

上述命令设置与说明如下：

➤ 设置 ResourceManager 主机与 NodeManager 的连接地址

NodeManager 通过这个地址向 ResourceManager 汇报运行情况。

```
<property>
  <name>yarn.resourcemanager.resource-tracker.address</name>
  <value>master:8025</value>
</property>
```

➤ 设置 ResourceManager 与 ApplicationMaster 的连接地址

ApplicationMaster 通过这个地址向 ResourceManager 申请资源、释放资源等。

```
<property>
  <name>yarn.resourcemanager.scheduler.address</name>
  <value>master:8030</value>
</property>
```

➤ 设置 ResourceManager 与客户端的连接地址

客户端通过该地址 ResourceManager 注册应用程序，删除应用程序等。

```
<property>
  <name>yarn.resourcemanager.address</name>
  <value>master:8050</value>
</property>
```

你可以在下列网址查看 YARN 架构图：

<http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html>

上述 yarn-site.xml 设置, 说明如图 5-16 所示。

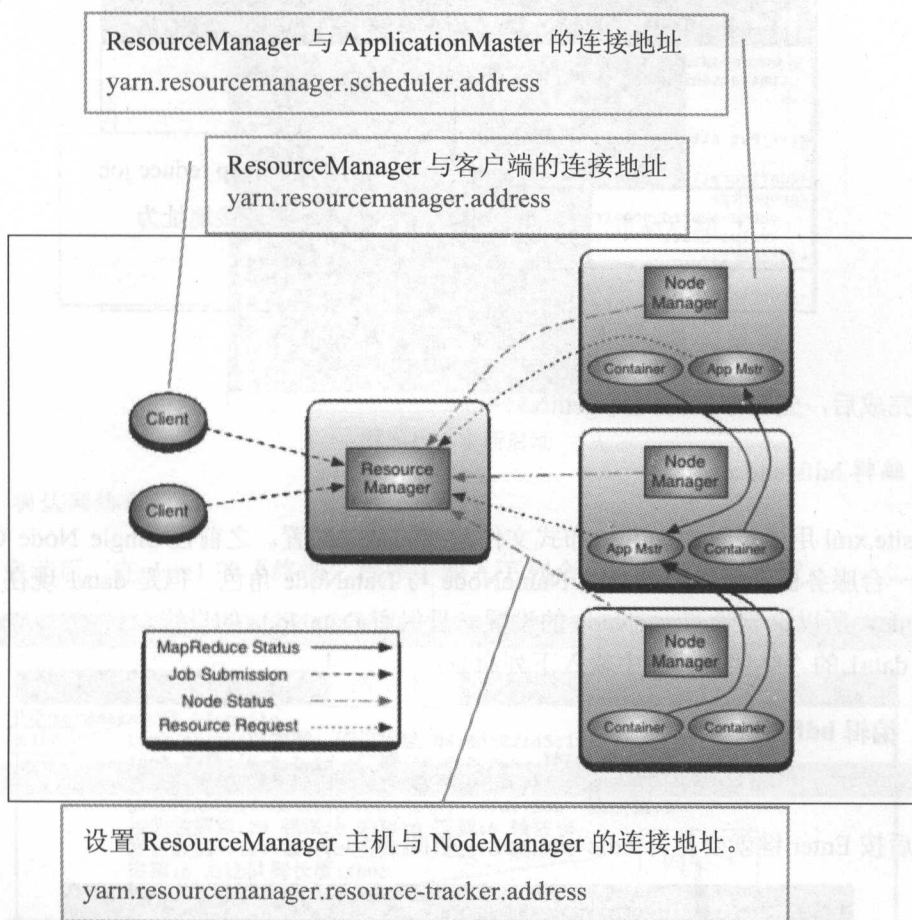


图 5-16 yarn-site.xml 的设置

步骤 07 编辑 mapred-site.xml

mapred-site.xml 用于设置监控 Map 与 Reduce 程序的 JobTracker 任务分配情况, 以及 TaskTracker 任务运行状况。在 data1 的“终端”程序中输入下列命令:

➤ 编辑 mapred-site.xml

```
sudo gedit /usr/local/hadoop/etc/hadoop/mapred-site.xml
```

输入后按 Enter 键就会打开 mapred-site.xml, 如图 5-17 所示。

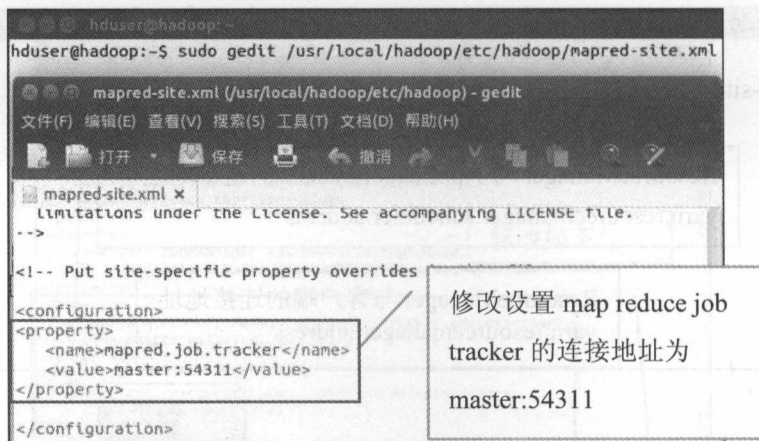


图 5-17 编辑 mapred-site.xml

编辑完成后，先保存，再关闭 gedit。

步骤 08 编辑 hdfs-site.xml

hdfs-site.xml 用于设置 HDFS 分布式文件系统的相关配置。之前在 Single Node Cluster 中因为只有一台服务器，所以同时身兼 NameNode 与 DataNode 角色。但是 data1 现在只是单纯的 DataNode，所以请删除 NameNode 的设置，只保留 DataNode 的设置。

请在 data1 的“终端”程序中输入下列命令：

➤ 编辑 hdfs-site.xml

```
sudo gedit /usr/local/hadoop/etc/hadoop/hdfs-site.xml
```

输入后按 Enter 键就会打开 hdfs-site.xml，如图 5-18 所示。

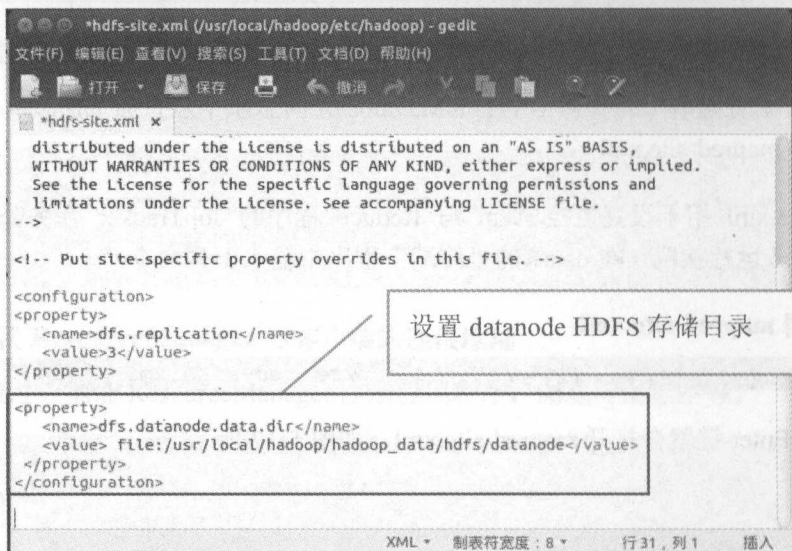


图 5-18 编辑 hdfs-site.xml 设置 datanode HDFS 存储目录

编辑完成后，先保存，再关闭 gedit。

步骤 09 data1 重新启动（见图 5-19）

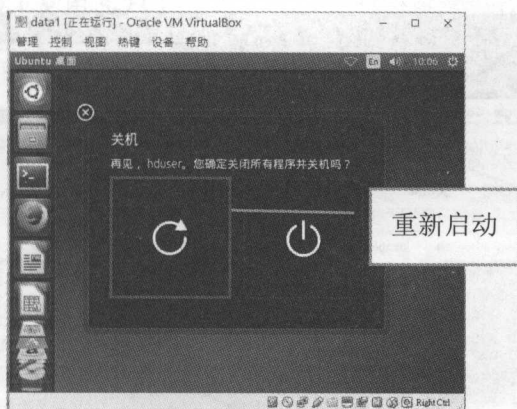


图 5-19 重新启动

步骤 10 确认网络设置

重新启动后，在 data1 的“终端”程序中输入下列命令，确认网络设置：

Ifconfig

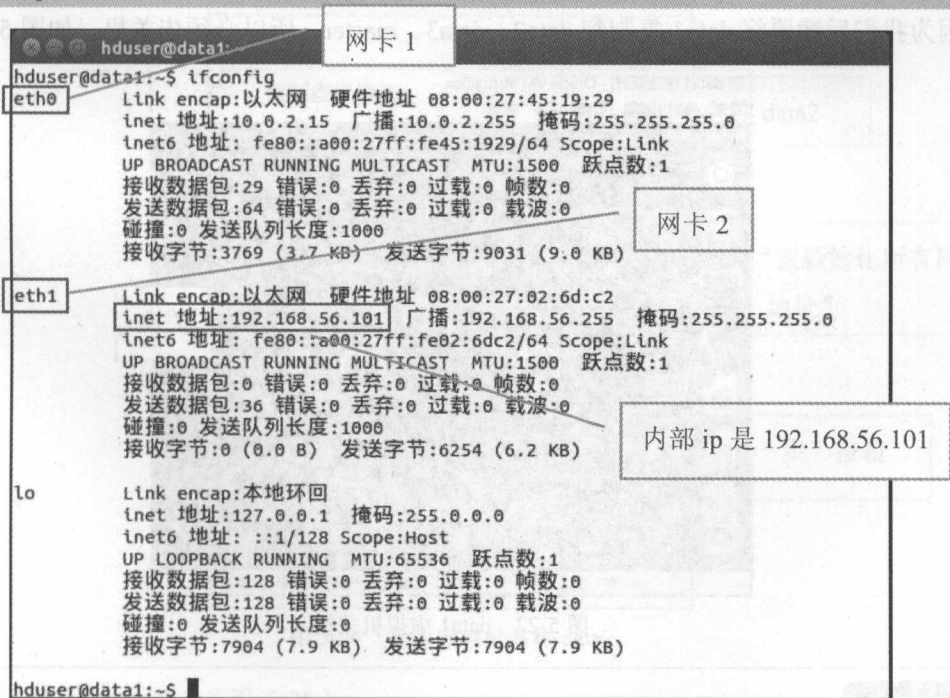


图 5-20 通过 ifconfig 命令确认网络配置

从以上界面（见图 15-20）可以看到，共有 2 张网卡 eth0 与 eth1，并且内部 ip 是 192.168.56.101。

步骤 11 确认对外网络连接正常

您可以打开浏览器，确认对外网络连接正常，如图 5-21 所示。

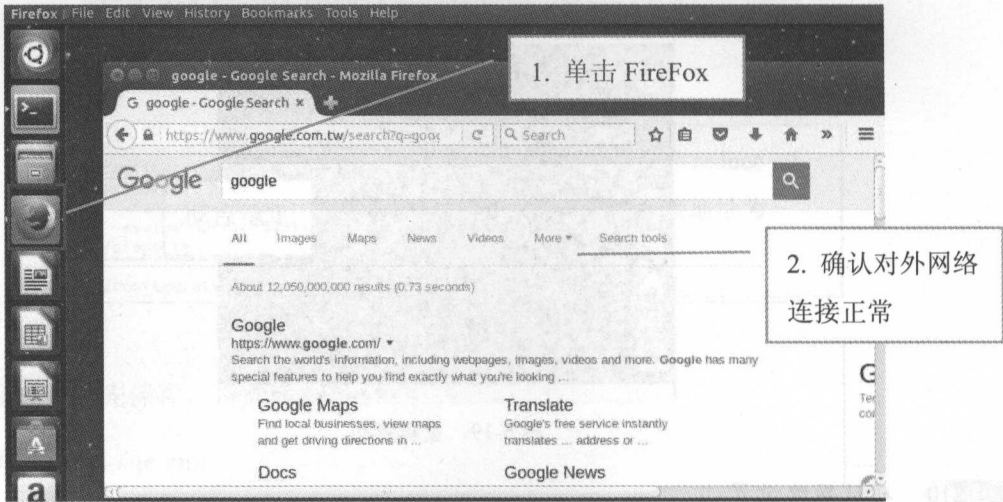


图 5-21 确认对外网络连接正常

步骤 12 data1 虚拟机关机

因为我们后续要将 data1 复制到 data2、data3、master，所以必须先关机，如图 5-22 所示。

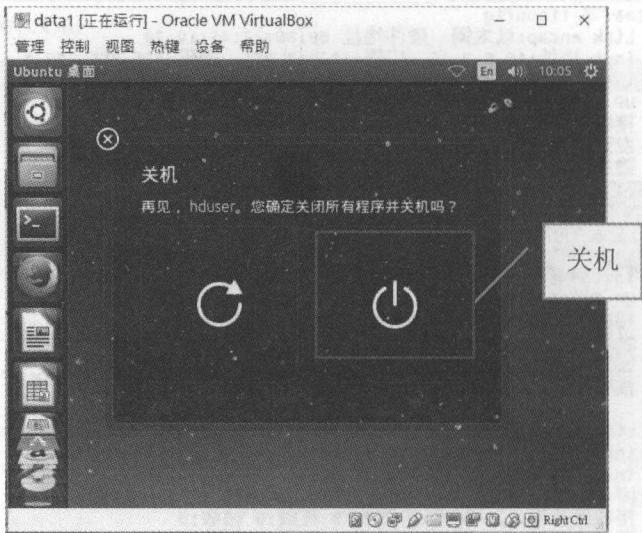


图 5-22 data1 虚拟机关机

5.4 复制 data1 服务器到 data2、data3、master

因为目前 data1 已经设置了 Hadoop Multi Node Cluster 共同的部分，为了节省安装时间，

所以复制 data1 到 data2、data3、master。如果你不使用虚拟机，也可以重复之前的步骤来创建 data2、data3、master。

步骤 01 data1 复制到 data2 (见图 5-23)

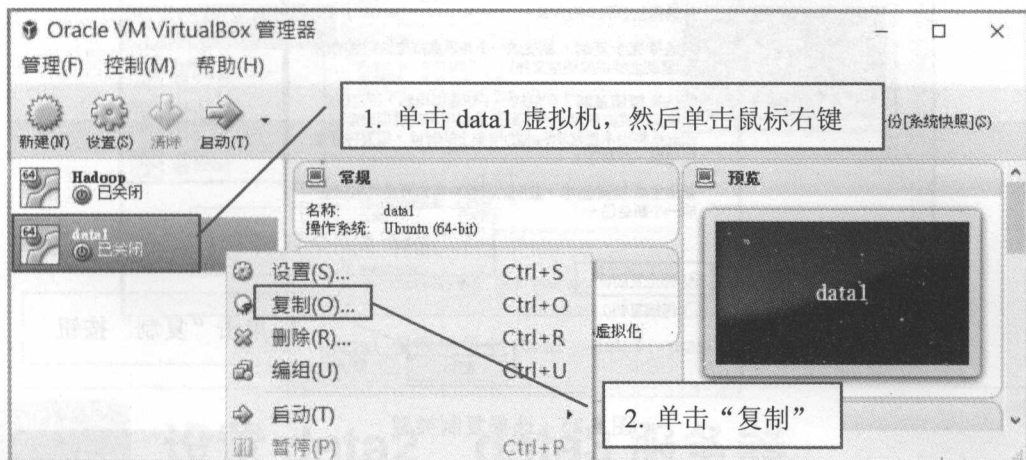


图 5-23 复制虚拟机 data1

步骤 02 输入虚拟机名称 data2 (见图 5-24)

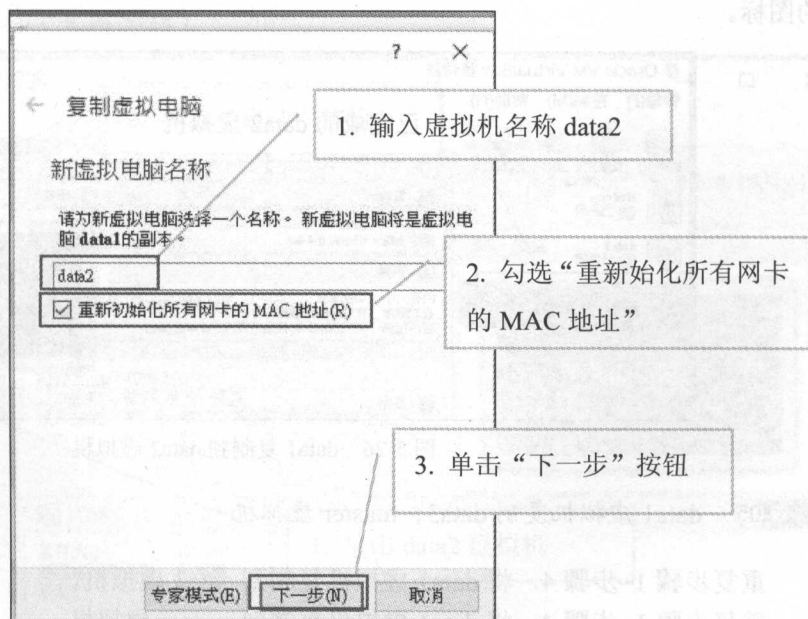


图 5-24 输入虚拟机名称 data2

步骤 03 选择复制类型 (见图 5-25)

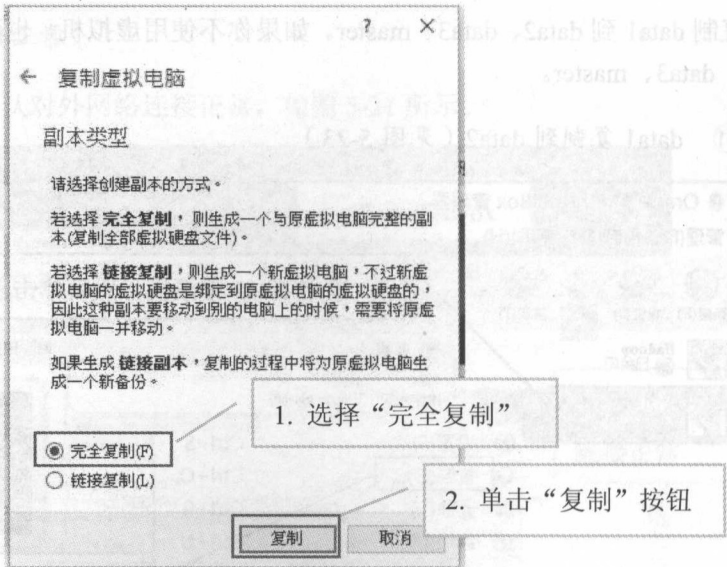


图 5-25 选择复制类型

步骤 04 data1 复制到 data2 虚拟机

如图 5-26 所示，单击“复制”按钮后，大约等候数分钟复制就完成了，则会出现 data2 的图标。

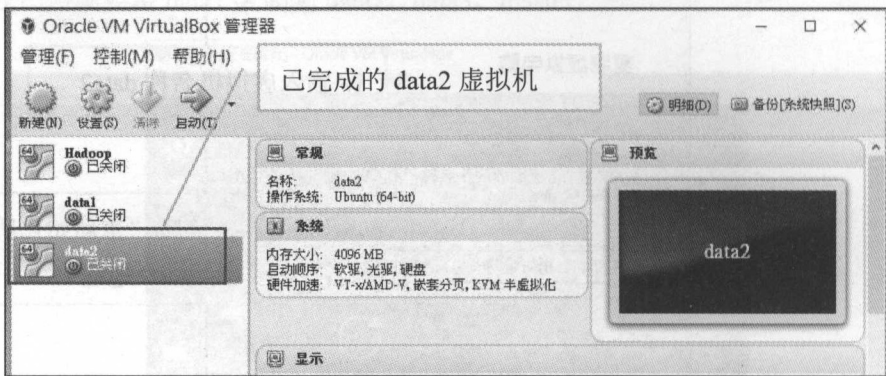


图 5-26 data1 复制到 data2 虚拟机

步骤 05 data1 虚拟机复制 data3，master 虚拟机

重复步骤 1~步骤 4，将 data1 虚拟机复制到 data3 虚拟机。

重复步骤 1~步骤 4，将 data1 虚拟机复制到 master 虚拟机。

如图 5-27 所示，我们已经将 data1 虚拟机复制到 data2、data3、master 虚拟机。

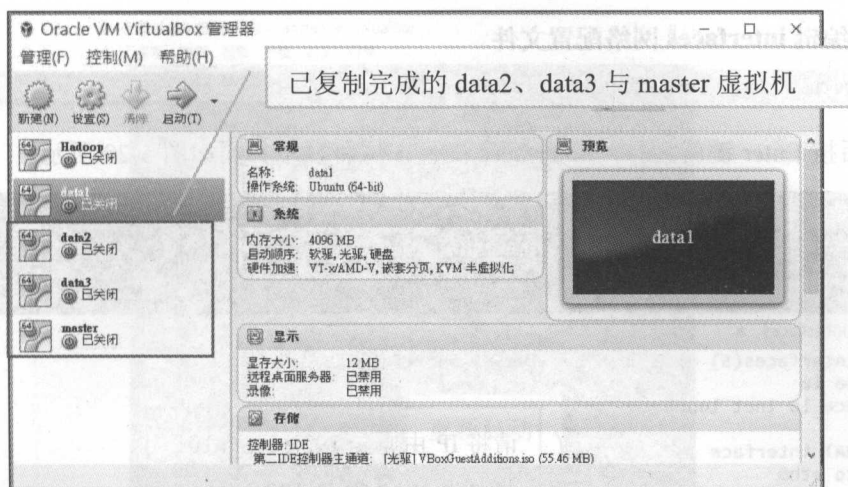


图 5-27 data1 虚拟机复制 data2、data3、master 虚拟机

5.5 设置 data2、data3 服务器

接下来，设置 data2、data3 固定 IP、hostname。

步骤 01 启动 data2 虚拟机（见图 5-28）

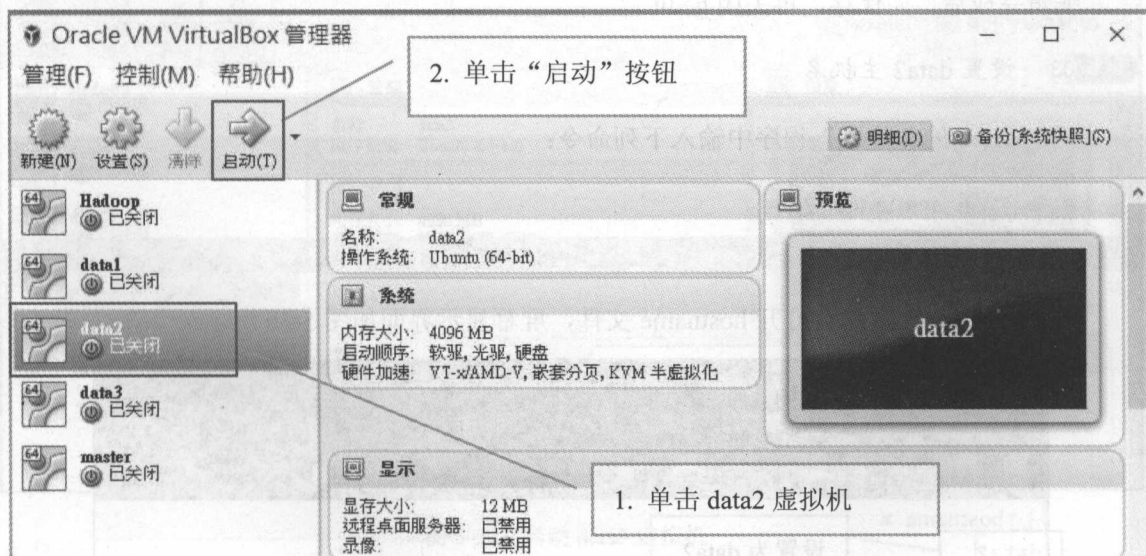


图 5-28 启动 data2 虚拟机

步骤 02 设置 data2 固定 IP 地址

我们必须设置 data2 虚拟主机每次开机都是使用固定 IP 地址：192.168.56.102。
在 data2 的“终端”程序中输入下列命令：

➤ 编辑 interfaces 网络配置文件

```
sudo gedit /etc/network/interfaces
```

输入后按 Enter 键就会打开 interfaces 文件，屏幕显示界面如图 5-29 所示。

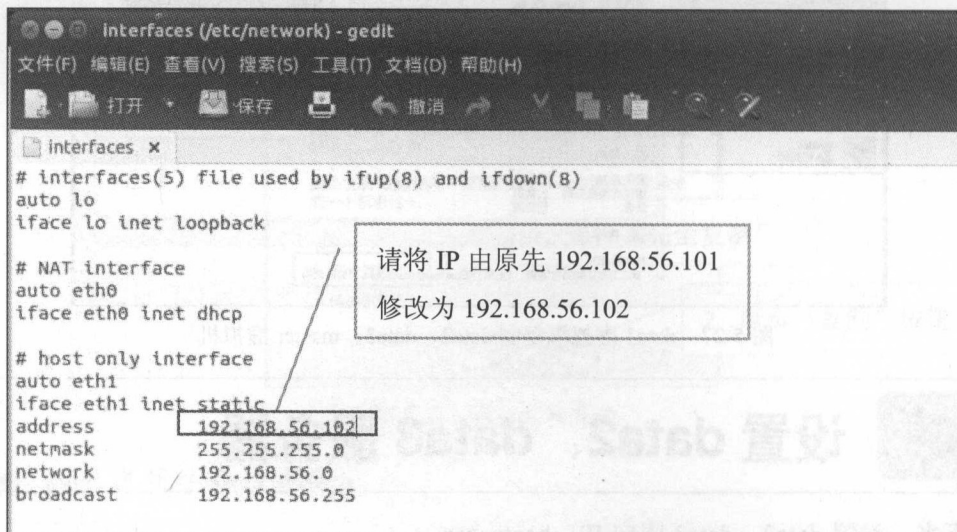


图 5-29 设置 data2 固定 IP 地址

编辑完成后，先保存，再关闭 gedit。

步骤 03 设置 data2 主机名

请在 data2 的“终端”程序中输入下列命令：

➤ 编辑 hostname 文件

```
sudo gedit /etc/hostname
```

输入后按 Enter 键就会打开 hostname 文件，屏幕显示界面如图 5-30 所示：

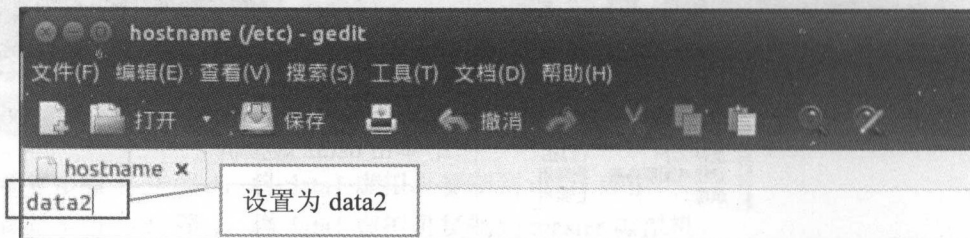


图 5-30 设置 data2 主机名

编辑完成后，先保存，再关闭 gedit。

步骤 04 data2 虚拟机关机（见图 5-31）

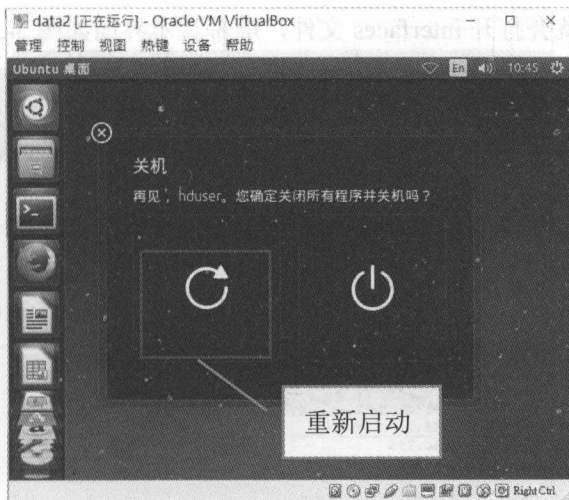


图 5-31 data2 虚拟机关机

步骤 05 启动 data3 虚拟机

启动 data3 虚拟机，如图 5-32 所示。接下来，设置 data3 固定 IP 地址、hostname。

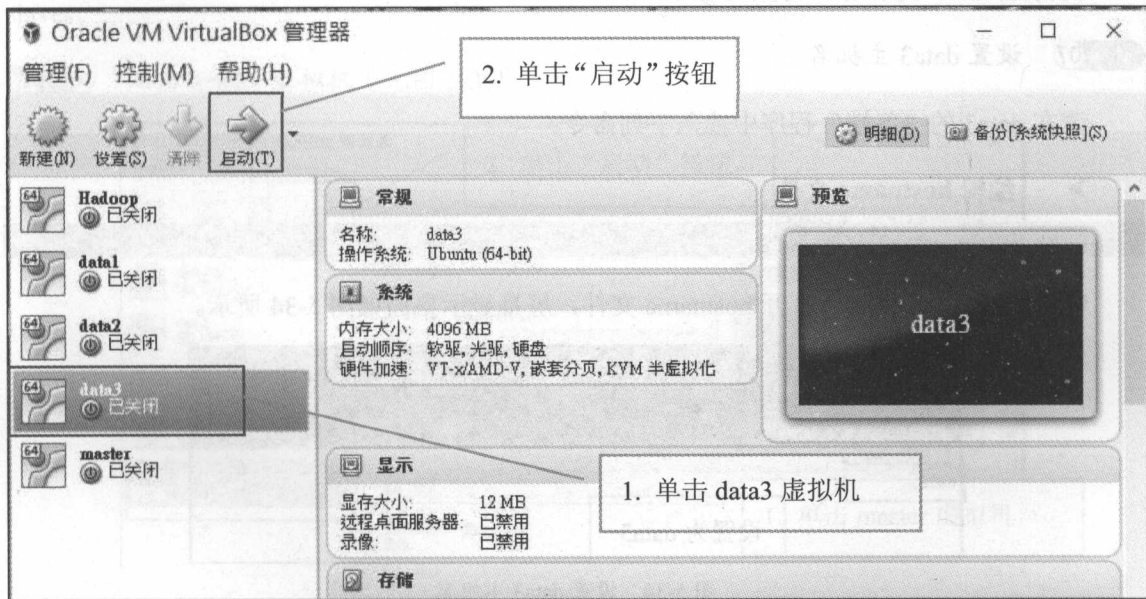


图 5-32 启动 data3 虚拟机

步骤 06 设置 data3 使用固定 IP 地址

接下来，我们要设置 data3 虚拟主机每次开机 IP 地址固定都是使用 192.168.56.103。请在 data3 的“终端”程序中输入下列命令：

➤ **编辑 interfaces 网络配置文件**

```
sudo gedit /etc/network/interfaces
```

输入后按 Enter 键就会打开 interfaces 文件，屏幕显示界面如图 5-33 所示。

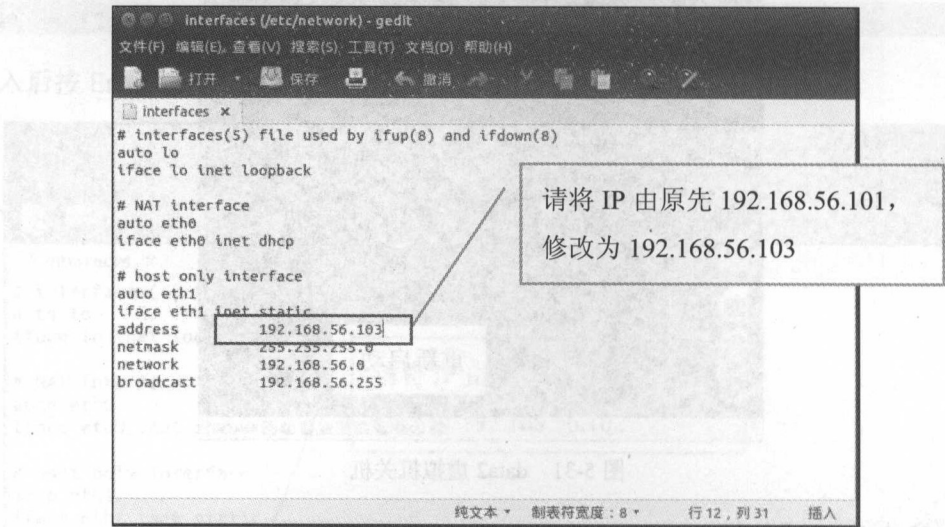


图 5-33 设置 data3 使用固定 IP 地址

编辑完成后，先保存，再关闭 gedit。

步骤 07 设置 data3 主机名

请在 data3 的“终端”程序中输入下列命令：

➤ 编辑 hostname 文件

```
sudo gedit /etc/hostname
```

输入后按 Enter 键就会打开 hostname 文件，屏幕显示界面如图 5-34 所示。

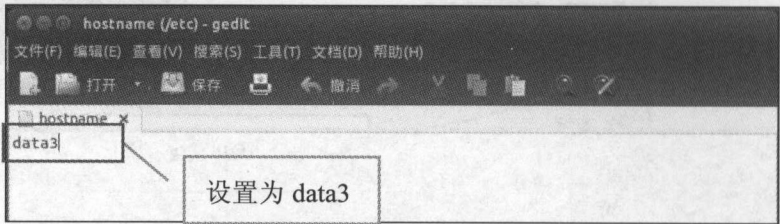


图 5-34 设置 data3 主机名

编辑完成后，先保存，再关闭 gedit。

步骤 08 data3 虚拟机关机（见图 5-35）

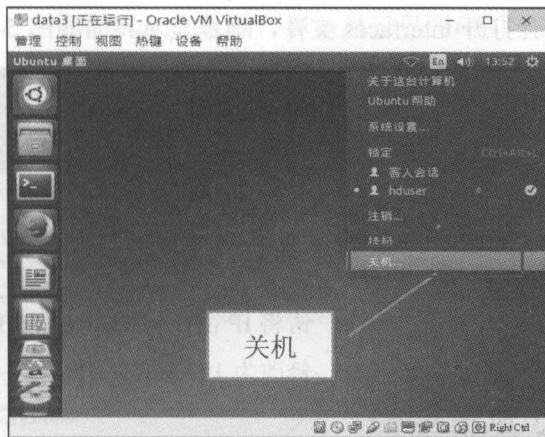


图 5-35 data3 虚拟机关机

5.6 设置 master 服务器

在 NameNode 服务器(master)中需要设置: 固定 IP 地址、hostname、hdfs-site.xml、masters、slaves。

步骤 01 启动 master 虚拟机 (见图 5-36)

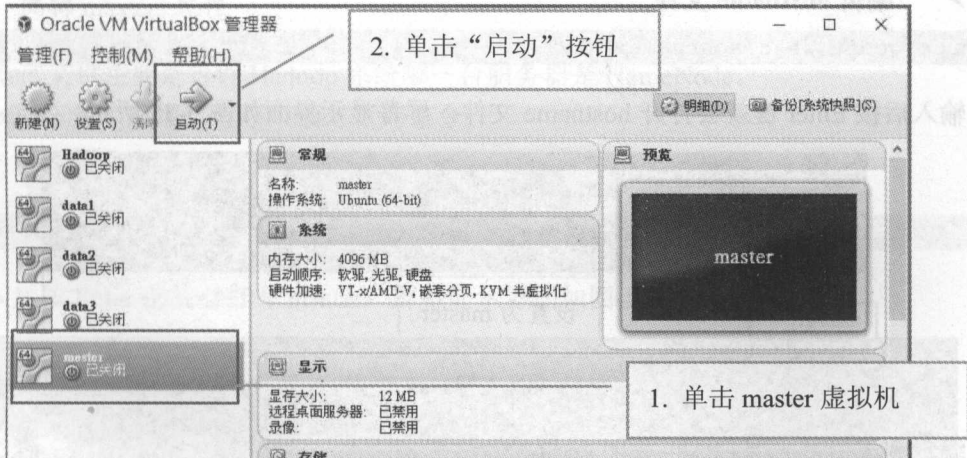


图 5-36 启动 mater 虚拟机

步骤 02 设置 master 固定 IP 地址

我们必须设置 master 虚拟主机每次开机都使用固定 IP 地址: 192.168.56.100。

在 master 的“终端”程序中输入下列命令:

➤ 编辑 master 的 interfaces 网络配置文件

```
sudo gedit /etc/network/interfaces
```

输入后按 Enter 键就会打开 interfaces 文件，屏幕显示界面如图 5-37 所示。

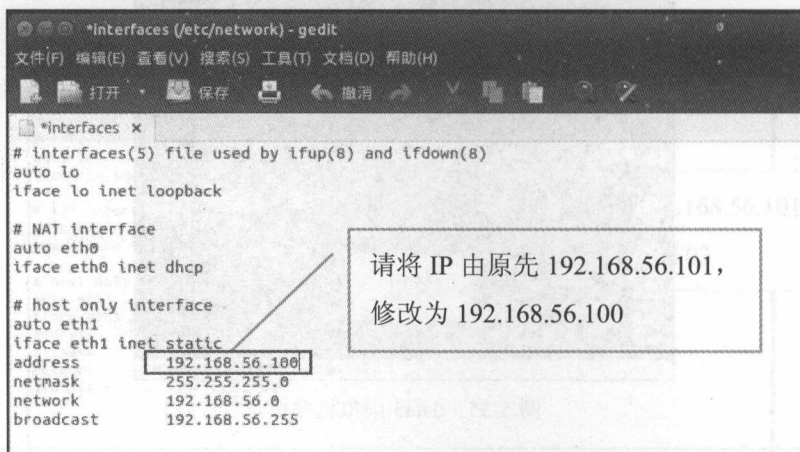


图 5-37 设置 master 固定 IP 地址

编辑完成后，先保存，再关闭 gedit。

步骤 03 设置 master 主机名

在 master 的“终端”程序中输入下列命令：

➤ 编辑 hostname 文件

```
sudo gedit /etc/hostname
```

输入后按 Enter 键就会打开 hostname 文件，屏幕显示界面如图 5-38 所示。

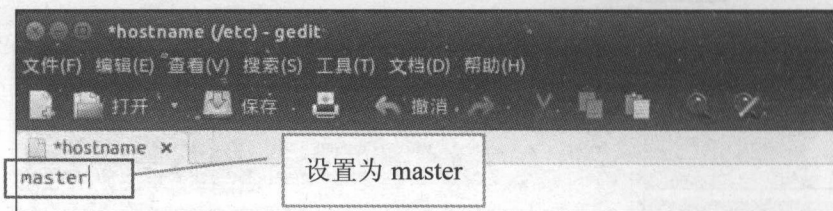


图 5-38 设置 master 主机名

编辑完成后，先保存，再关闭 gedit。

步骤 04 设置 hdfs-site.xml

hdfs-site.xml 用于设置 HDFS 分布式文件系统相关配置的设置。因为 master 现在只是单纯的 NameNode，请删除 DataNode 的 HDFS 设置，并加入 NameNode 的 HDFS 设置。在 master 的“终端”程序中输入下列命令：

➤ 编辑 hdfs-site.xml

```
sudo gedit /usr/local/hadoop/etc/hadoop/hdfs-site.xml
```

输入后按 Enter 键就会打开 hdfs-site.xml，屏幕显示界面如图 5-39 所示。

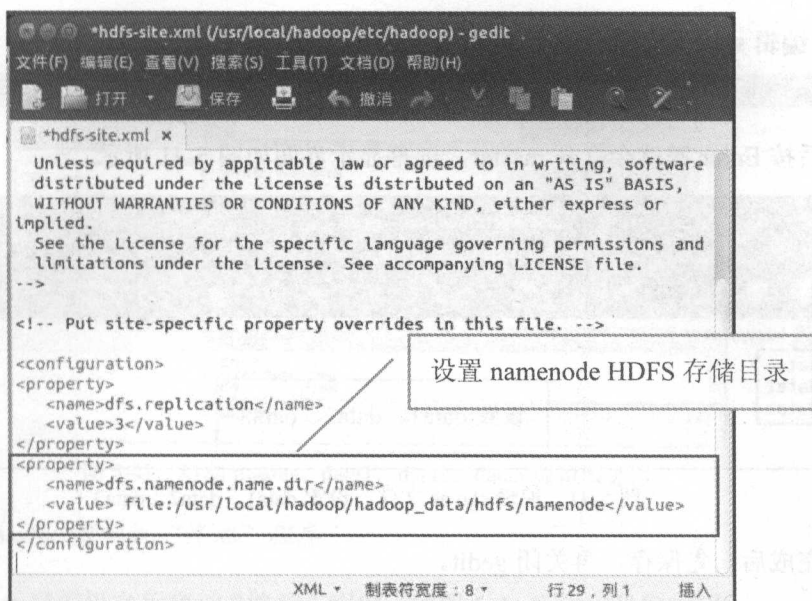


图 5-39 编辑 hdfs-site.xml，设置 namenode HDFS 存储目录

编辑完成后，先保存，再关闭 gedit。

步骤 05 编辑 masters 文件

masters 文件主要是告诉 Hadoop 系统哪一台服务器是 NameNode。
在 master 的“终端”程序中输入下列命令：

➤ 编辑 master 文件

```
sudo gedit /usr/local/hadoop/etc/hadoop/masters
```

输入后按 Enter 键就会打开 master，屏幕显示界面如图 5-40 所示。

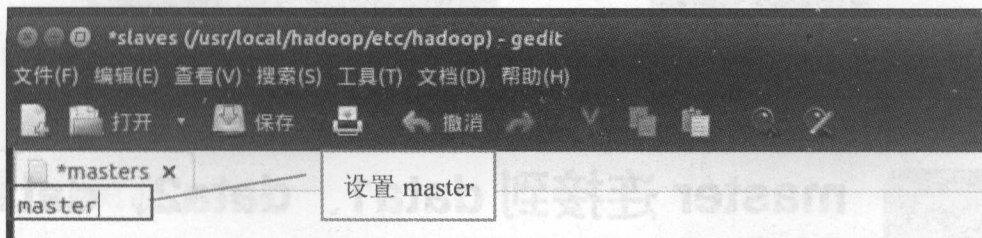


图 5-40 编辑 masters 文件，设置 master

编辑完成后，先保存，再关闭 gedit。

步骤 06 编辑 slaves 文件

slaves 文件主要是告诉 Hadoop 系统哪些服务器是 DataNode。

在 master 的“终端”程序中输入下列命令：

➤ 编辑 slaves 文件

```
sudo gedit /usr/local/hadoop/etc/hadoop/slaves
```

输入后按 Enter 键就会打开 master，屏幕显示界面如图 5-41 所示。

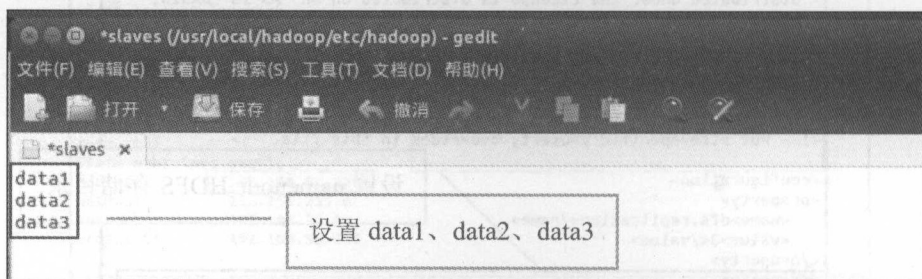


图 5-41 编辑 slaves 文件，设置 data1、data2、data3

编辑完成后，先保存，再关闭 gedit。

步骤 07 master 虚拟机关机

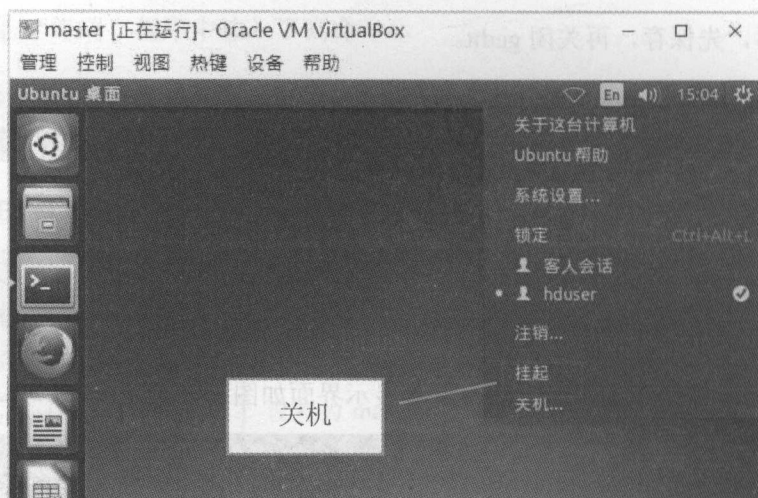


图 5-42 master 虚拟机关机

5.7

master 连接到 data1、data2、data3 创建 HDFS 目录

之前的步骤我们已经创建了 master 与 data1、data2、data3 服务器。接下来，要创建 NameNode (master) 的 SSH 连接到 DataNode (data1、data2、data3)，并创建 HDFS 相关目录。

步骤 01 启动 master、data1、data2、data3

首先必须要启动所有虚拟服务器 master、data1、data2、data3，如图 5-43 所示。

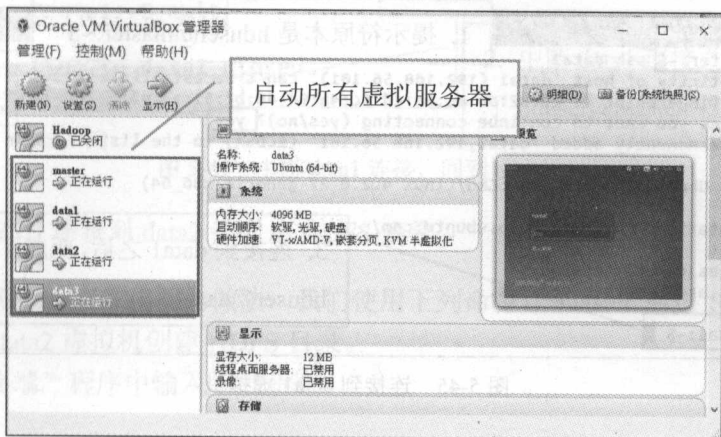


图 5-43 启动 master、data1、data2、data3 虚拟服务器

步骤 02 启动 master 虚拟机“终端”程序

切换到 master 虚拟机之后启动“终端”程序，如图 5-44 所示。我们将用 master 的“终端”程序通过 SSH 把 master 连接到 data1 服务器。

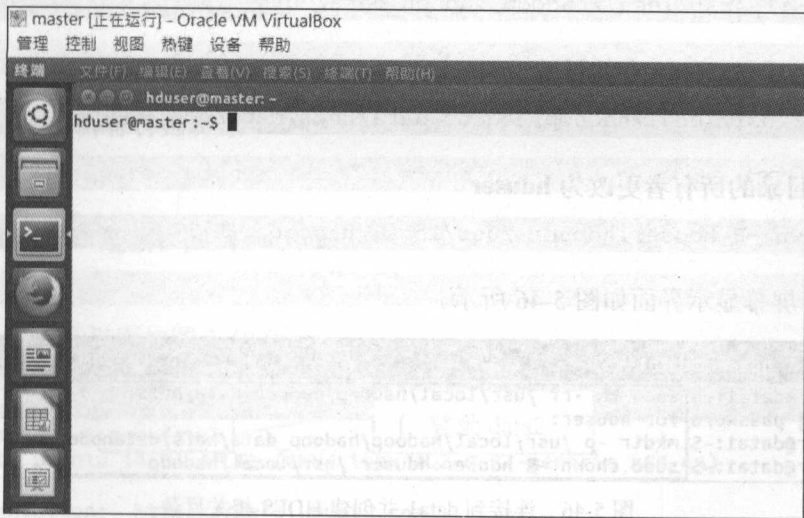


图 5-44 启动 master 虚拟机的“终端”程序

步骤 03 连接到 data1 虚拟机

在 master 的“终端”程序中输入下列命令：

➤ master 通过 SSH 连接到 data1 虚拟机

```
ssh data1
```

如图 5-45 所示，输入命令后按 Enter 键，就会连接到 data1。请注意！提示符原本是

hduser@master:~\$, 连接到 data1 之后变成了 hduser@data1:~\$, 代表已经成功连接 data1。

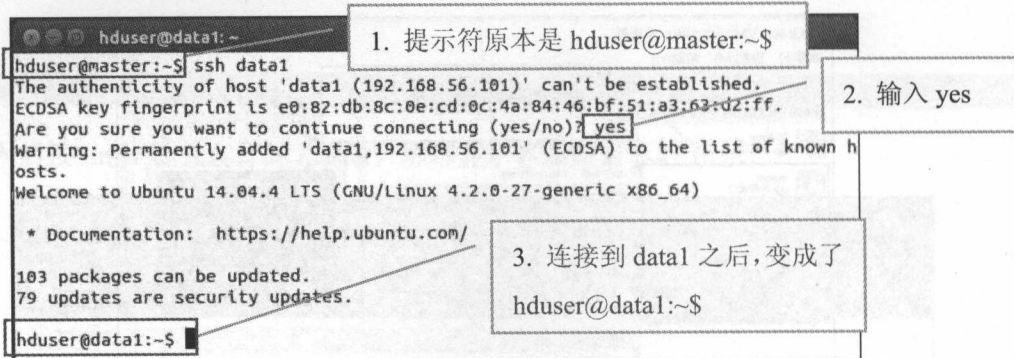


图 5-45 连接到 data1 虚拟机

步骤 04 连接到 data1 创建 HDFS 相关目录

登录 data1 后, 我们将在 data1 创建 HDFS 相关目录如下。

在 master 的“终端”程序中输入下列命令:

➤ 删除 hdfs 所有目录

```
sudo rm -rf /usr/local/hadoop/hadoop_data/hdfs
```

➤ 创建 DataNode 存储目录

```
mkdir -p /usr/local/hadoop/hadoop_data/hdfs/datanode
```

➤ 将目录的所有者更改为 hduser

```
sudo chown -R hduser:hduser /usr/local/hadoop
```

完成后, 屏幕显示界面如图 5-46 所示。

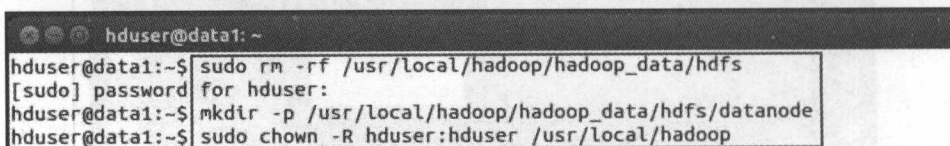


图 5-46 连接到 data1 并创建 HDFS 相关目录

步骤 05 中断 data1 连接, 回到 master

完成后, 在 master 的“终端”中输入下列命令:

➤ 中断 data1 连接, 回到 master

```
exit
```

如图 5-47 所示, 运行 exit 之后, 原本提示符是 hduser@data1:~\$, 中断 data1 连接后恢复成 hduser@master:~\$, 代表连接中断回到 master 虚拟机。

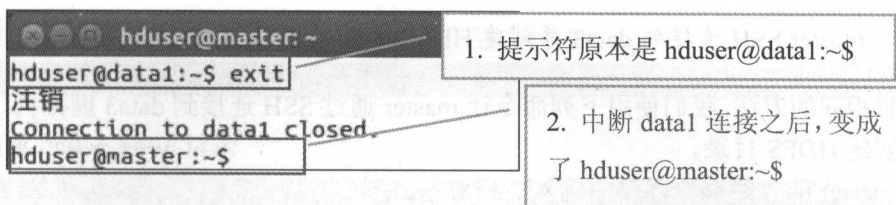


图 5-47 中断 data1 连接, 回到 master

步骤 06 master SSH 连接到 data2 并创建 HDFS 目录

接下来, 参照上一小节相同的做法, 我们使用下列命令让 master 通过 SSH 连接到 data2 虚拟机, 并且在 data2 虚拟机创建 HDFS 目录。

在 master “终端” 程序中输入下列命令:

➤ **master 通过 SSH 连接到 data2 虚拟机**

```
ssh data2
```

➤ **删除 hdfs 所有目录**

```
sudo rm -rf /usr/local/hadoop/hadoop_data/hdfs
```

➤ **创建 DataNode 存储目录**

```
mkdir -p /usr/local/hadoop/hadoop_data/hdfs/datanode
```

➤ **将目录的所有者更改为 hduser**

```
sudo chown -R hduser:hduser /usr/local/hadoop
```

➤ **中断 data2 连接, 回到 master**

```
exit
```

完成后屏幕显示界面如图 5-48 所示。

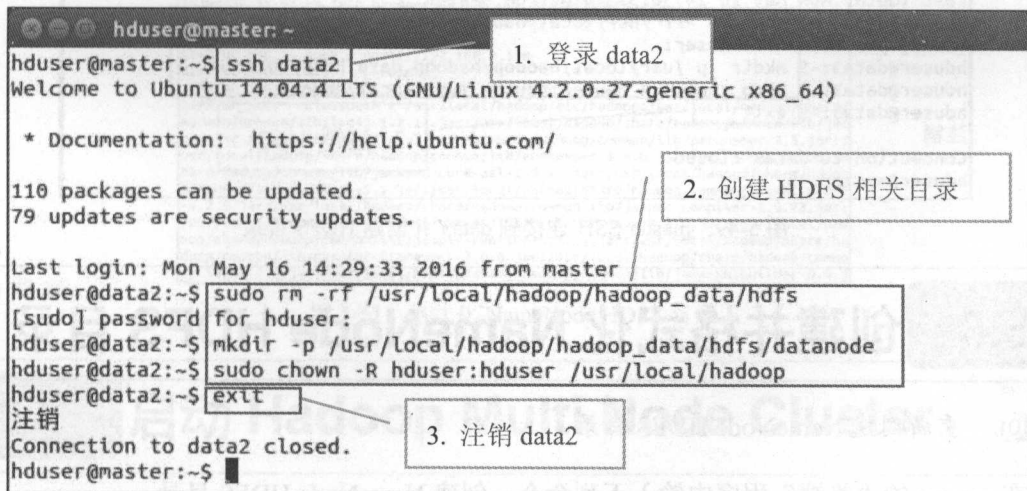


图 5-48 master SSH 连接到 data2 并创建 HDFS 目录

步骤 07 master SSH 连接到 data3 并创建 HDFS 目录

参照相同的方法,我们使用下列命令让 master 通过 SSH 连接到 data3 虚拟机,并且在 data3 虚拟机创建 HDFS 目录。

在 master 的“终端”程序中输入下列命令:

➤ **master 通过 SSH 连接到 data3 虚拟机**

```
ssh data3
```

➤ **删除 hdfs 所有目录**

```
sudo rm -rf /usr/local/hadoop/hadoop_data/hdfs
```

➤ **创建 DataNode 存储目录**

```
mkdir -p /usr/local/hadoop/hadoop_data/hdfs/datanode
```

➤ **将目录的所有者更改为 hduser**

```
sudo chown -R hduser:hduser /usr/local/hadoop
```

➤ **中断 data3 连接, 回到 master**

```
exit
```

完成后屏幕显示界面如图 5-49 所示。

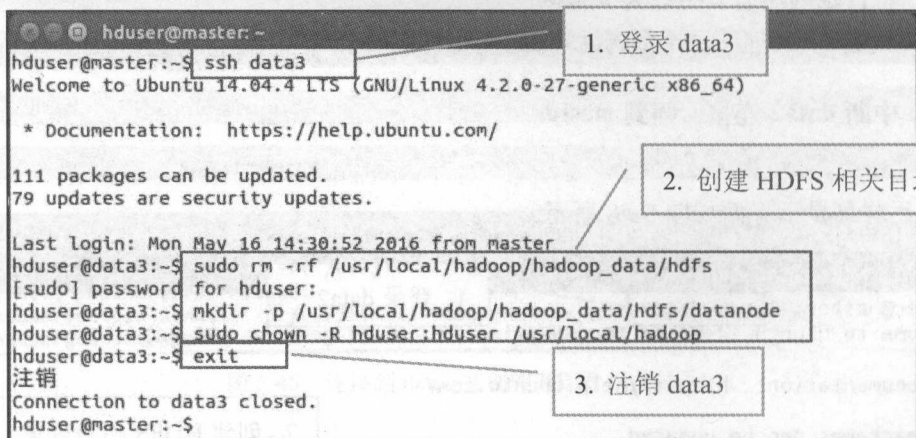


图 5-49 master SSH 连接到 data3 并创建 HDFS 目录

5.8 创建并格式化 NameNode HDFS 目录

步骤 01 重新创建 NameNode HDFS 目录

在 master 的“终端”程序中输入下列命令, 创建 NameNode HDFS 目录:

➤ 删除之前的 HDFS 目录

```
sudo rm -rf /usr/local/hadoop/hadoop_data/hdfs
```

➤ 创建 NameNode 目录

```
mkdir -p /usr/local/hadoop/hadoop_data/hdfs/namenode
```

➤ 将目录的所有者更改为 hduser

```
sudo chown -R hduser:hduser /usr/local/hadoop
```

在 master 虚拟机的“终端”程序中输入如图 5-50 所示的命令。

```
hduser@master:~$ sudo rm -rf /usr/local/hadoop/hadoop_data/hdfs
hduser@master:~$ mkdir -p /usr/local/hadoop/hadoop_data/hdfs/namenode
hduser@master:~$ sudo chown -R hduser:hduser /usr/local/hadoop
hduser@master:~$
```

图 5-50 删除并重新创建 NameNode HDFS 目录

步骤 02 格式化 NameNode HDFS 目录

之前我们已经创建了 DataNode 与 NameNode 的 HDFS 目录。接下来，需要格式化 HDFS 目录。在 master 的“终端”程序中输入下列命令：

➤ 格式化 NameNode HDFS 目录

```
hadoop namenode -format
```

格式化时屏幕显示界面如图 5-51 所示。

```
hduser@master:~$ hadoop namenode -format
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.

15/04/29 11:07:03 INFO namenode.NameNode: STARTUP_MSG:
/*****
STARTUP_MSG: Starting NameNode
STARTUP_MSG: host = master/192.168.0.100
STARTUP_MSG: args = [-format]
STARTUP_MSG: version = 2.6.0
STARTUP_MSG: classpath = /usr/local/hadoop/etc/hadoop:/usr/local/hadoop/share/
hadoop/common/lib/log4j-1.2.17.jar:/usr/local/hadoop/share/hadoop/common/lib/jac
kson-xc-1.9.13.jar:/usr/local/hadoop/share/hadoop/common/lib/paranamer-2.3.jar:/
usr/local/hadoop/share/hadoop/common/lib/zookeeper-3.4.6.jar:/usr/local/hadoop/s
hare/hadoop/common/lib/jackson-core-asl-1.9.13.jar:/usr/local/hadoop/share/hadoo
p/common/lib/jettison-1.1.jar:/usr/local/hadoop/share/hadoop/common/lib/httpcore
-4.2.5.jar:/usr/local/hadoop/share/hadoop/common/lib/jasper-compiler-5.5.23.jar:/
usr/local/hadoop/share/hadoop/common/lib/protobuf-java-2.5.0.jar:/usr/local/had
oop/share/hadoop/common/lib/jasper-runtime-5.5.23.jar:/usr/local/hadoop/share/ha
dooop/common/lib/curator-framework-2.6.0.jar:/usr/local/hadoop/share/hadoop/commo
n/lib/xz-1.0.jar:/usr/local/hadoop/share/hadoop/common/lib/java-xmlbuilder-0.4.3
```

图 5-51 格式化 NameNode HDFS 目录

5.9 启动 Hadoop Multi Node Cluster

步骤 01 启动 Hadoop Multi Node Cluster

到目前为止我们已经完成 Hadoop cluster 的构建，可以在 master 的“终端”程序中输入下列命令开始操作：

➤ 分别启动 HDFS 与 YARN

命令	说明
start-dfs.sh	启动 HDFS
start-yarn.sh	启动 Hadoop MapReduce 框架 Yarn

或

➤ 同时启动 HDFS 与 YARN

命令	说明
start-all.sh	同时启动 HDFS 与 YARN

如图 5-52 所示。

```
hduser@master: ~
hduser@master:~$ start-all.sh
This script is Deprecated. Instead use start-dfs.sh and start-yarn.sh
Starting namenodes on [master]
master: starting namenode, logging to /usr/local/hadoop/logs/hadoop-hduser-namenode-master.out
data3: starting datanode, logging to /usr/local/hadoop/logs/hadoop-hduser-datano
de-data3.out
data1: starting datanode, logging to /usr/local/hadoop/logs/hadoop-hduser-datano
de-data1.out
data2: starting datanode, logging to /usr/local/hadoop/logs/hadoop-hduser-datano
de-data2.out
Starting secondary namenodes [0.0.0.0]
0.0.0.0: starting secondarynamenode, logging to /usr/local/hadoop/loqs/hadoop-hd
user-secondarynamenode-master.out
starting yarn daemons
starting resourcemanager, logging to /usr/local/hadoop/logs/yarn-hduser-resource
manager-master.out
data2: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-hduser-nodem
anager-data2.out
data3: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-hduser-nodem
anager-data3.out
data1: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-hduser-nodem
anager-data1.out
hduser@master:~$
```

1. 启动 HDFS

2. 启动 YARN

图 5-52 启动 Hadoop Multi Node Cluster 的 HDFS 和 YARN

步骤 02 查看 master (NameNode) 进程 (process)

jps (Java Virtual Machine Process Status Tool)，可以用于查看当前所运行的进程，在 master 的“终端”程序中输入下列命令：

➤ 查看 master (NameNode) 的进程 (process) 是否启动

```
jps
```

运行后屏幕显示界面如图 5-53 所示，可以看见 master 服务器的状态。

- **HDFS 功能:** Namenode、Secondary NameNode 已经启动。
- **MapReduce2 (YARN):** Resource Manager 已经启动。

```

hduser@master:~$ jps
4646 Jps
4350 ResourceManager
3998 NameNode
4185 SecondaryNameNode
  
```

图 5-53 查看 master (NameNode) 进程

步骤 03 查看 data1 (DataNode) 进程

在 master 的“终端”程序中输入下列命令，通过 SSH 连接到 data1 来查看 data1 (DataNode) 进程。

➤ master 通过 SSH 连接到 data1 虚拟机

```
ssh data1
```

➤ 查看 data1 (DataNode) 所运行的进程

```
jps
```

➤ 注销 data1 回到 master

```
exit
```

运行后屏幕显示界面如图 5-54 所示，从中可以看见 data1 服务器的状态：

- **HDFS 功能:** DataNode 已经启动。
- **MapReduce2 (YARN):** Node Manager 已经启动。

```

hduser@master:~$ ssh data1
Welcome to Ubuntu 14.04.4 LTS (GNU/Linux 4.2.0-27-generic x86_64)

 * Documentation:  https://help.ubuntu.com/
110 packages can be updated.
79 updates are security updates.

Last login: Sun May 15 15:13:06 2016 from master
hduser@data1:~$ jps
1956 NodeManager
1872 DataNode
2044 Jps
hduser@data1:~$ exit
注销
Connection to data1 closed.
hduser@master:~$
  
```

图 5-54 查看 data1 (DataNode) 进程

也可以使用相同方式连接到 data2、data3 进程。

5.10

打开 Hadoop ResourceManager Web 界面

Hadoop ResourceManager Web 界面可用于查看当前 Hadoop 的状态：Node 节点、应用程序、进程运行情况。

步骤 01 打开 ResourceManager Web 界面

打开浏览器在地址栏输入：

➤ ResourceManager Web 界面网址

http://master:8088/

参照下列步骤，就可以看到如图 5-55 所示的屏幕显示界面。

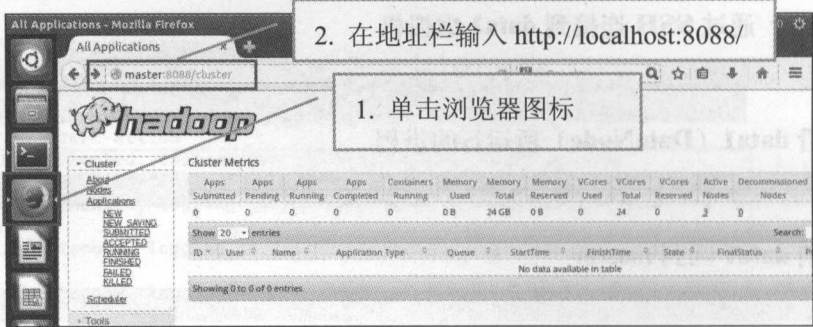


图 5-55 打开 Hadoop ResourceManager Web 界面

步骤 02 查看已经运行的节点 Nodes

当单击 Nodes 时，会显示当前的节点。当前共有 3 个节点 Nodes：data1、data2、data3，如图 5-56 所示（单击 Nodes 时，有时尚未出现节点，可以刷新几次就会出现）。

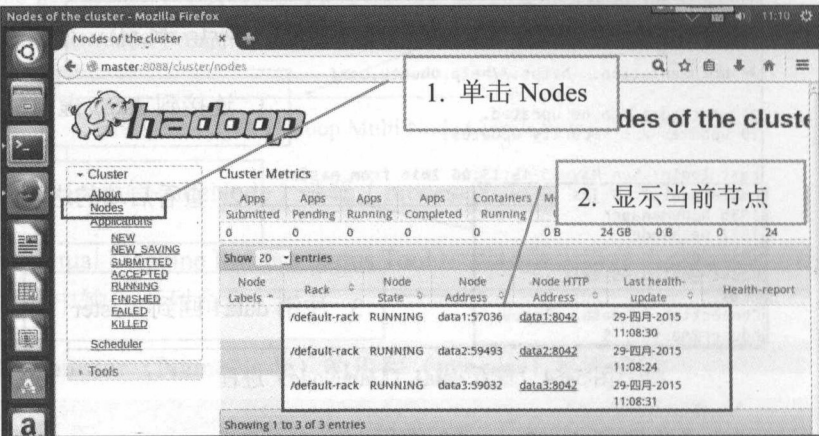


图 5-56 查看已经运行的节点 Nodes

5.11 打开 NameNode Web 界面

HDFS Web 界面可用于检查当前的 HDFS 与 DataNode 运行的情况。

步骤 01 打开 NameNode HDFS Web 用户界面

打开浏览器在地址栏输入：

➤ **HDFS Web 用户界面网址**

http://master:50070/

可以看到屏幕显示界面如图 5-57 所示。

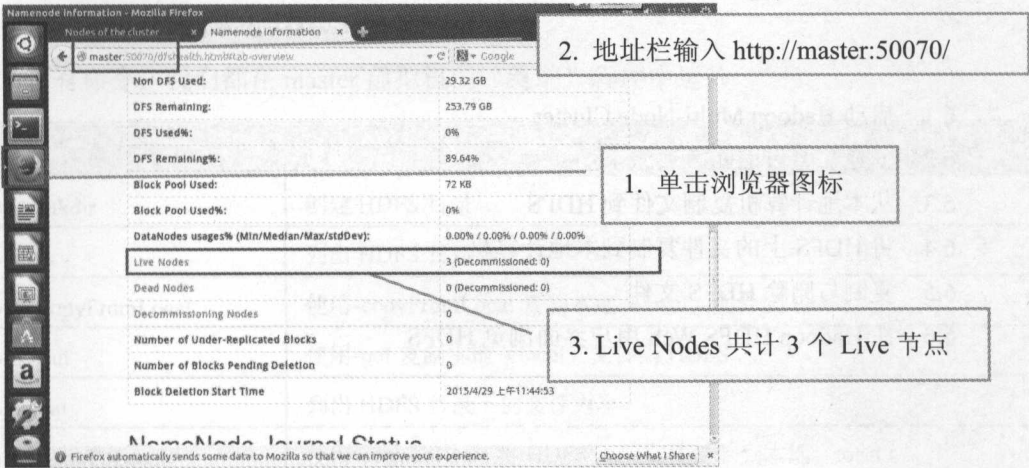


图 5-57 打开 NameNode HDFS Web 用户界面

步骤 02 查看 Datanodes

单击 Datanodes，可以看到当前启动了 3 个节点 Datanodes，如图 5-58 所示。

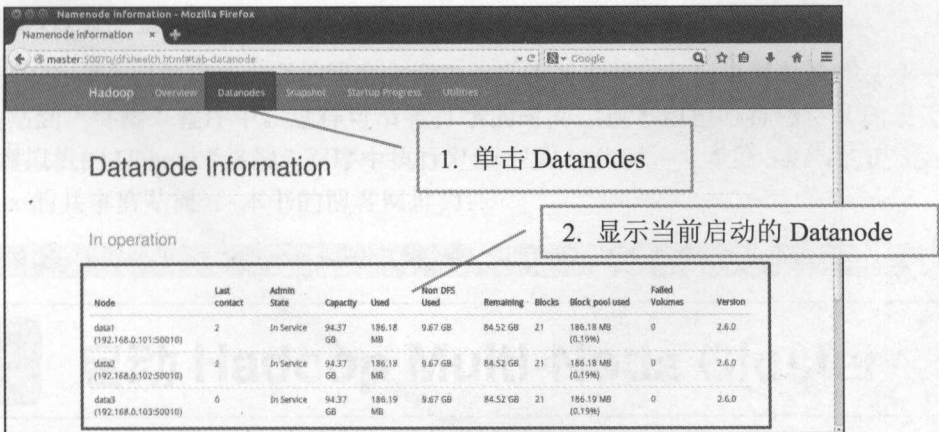
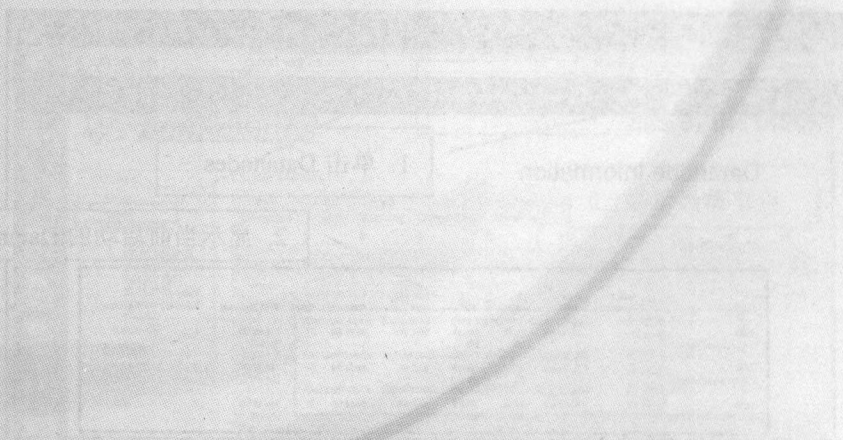


图 5-58 查看 Datanodes

第 6 章

Hadoop HDFS命令

- 6.1 启动 Hadoop Multi-Node Cluster
- 6.2 创建与查看 HDFS 目录
- 6.3 从本地计算机复制文件到 HDFS
- 6.4 将 HDFS 上的文件复制到本地计算机
- 6.5 复制与删除 HDFS 文件
- 6.6 在 Hadoop HDFS Web 用户界面浏览 HDFS



在第 1 章我们已经介绍了 HDFS 概念，本章中将介绍可以在“终端”程序中使用的 HDFS 命令，对 HDFS 进行操作，示意图如图 6-1 所示。HDFS 命令格式如下：

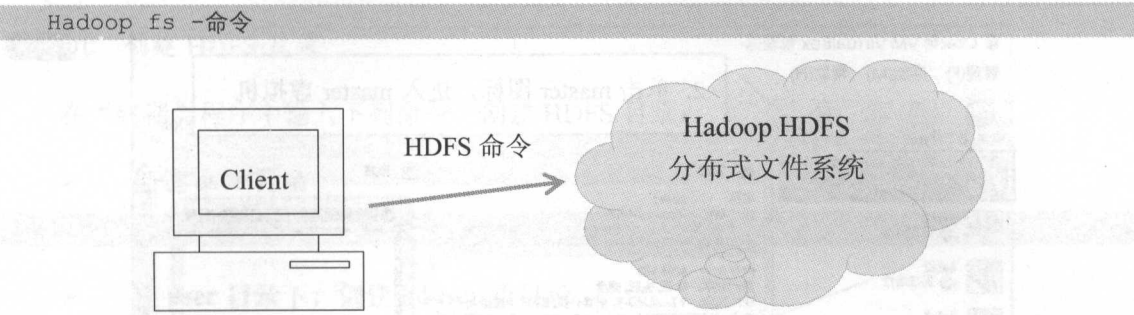


图 6-1 HDFS 命令对 HDFS 进行操作

本章介绍下列常用 HDFS 命令

本章所有命令，我们都在 master 虚拟机的“终端”程序中运行。

命令	说明
hadoop fs -mkdir	创建 HDFS 目录
hadoop fs -ls	列出 HDFS 目录
hadoop fs -copyFromLocal	使用-copyFromLocal 复制本地（local）文件到 HDFS
hadoop fs -put	使用-put 复制本地（local）文件到 HDFS
hadoop fs -cat	列出 HDFS 目录下的文件内容
hadoop fs -copyToLocal	使用-copyToLocal 将 HDFS 上的文件复制到本地（local）
hadoop fs -get	使用-get 将 HDFS 上的文件复制到本地（local）
hadoop fs -cp	复制 HDFS 文件
hadoop fs -rm	删除 HDFS 文件

HDFS 命令整理

以下 HDFS 命令已整理在本书的博客文章中。当练习安装时，可以复制博客文章中的命令，然后粘贴到“终端”程序中。这样可节省打字的时间，也不用担心打错字（如果无法在 VirtualBox 虚拟机的 Ubuntu “终端”程序中执行复制/粘贴操作时，参考第 3.9 节的说明，设置好 VirtualBox 的共享剪贴板）。本书的博客网址为：

<http://blog.sina.com.cn/hadoopsparkbook>

6.1 启动 Hadoop Multi-Node Cluster

在示范 HDFS 命令之前，我们必须先启动 Hadoop Multi-Node Cluster。

步骤 01 启动所有虚拟服务器

首先必须要启动所有虚拟服务器 master、data1、data2、data3，如图 6-2 所示。

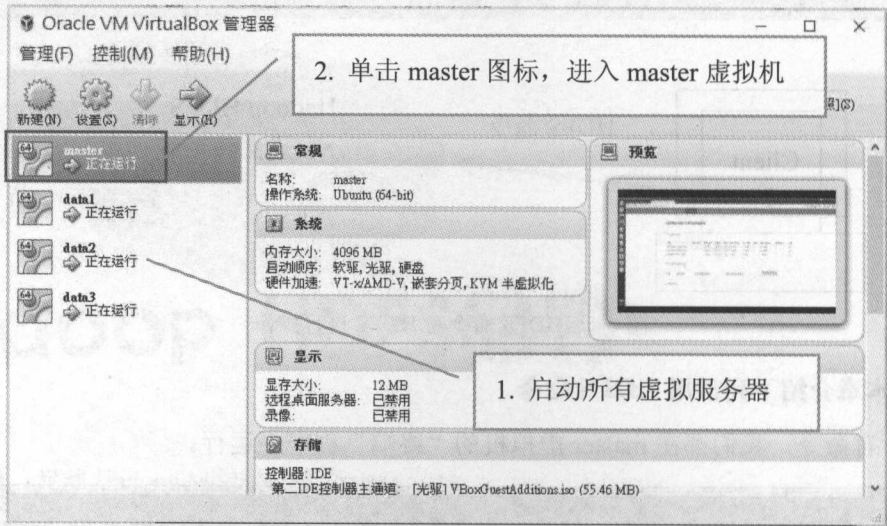


图 6-2 启动所有虚拟服务器

步骤 02 start-all.sh 启动 Hadoop Multi-Node Cluster

在 master 虚拟机启动“终端”程序，输入下列命令：

➤ 启动 Hadoop Multi-Node Cluster

start-all.sh

运行后屏幕显示界面如图 6-3 所示。

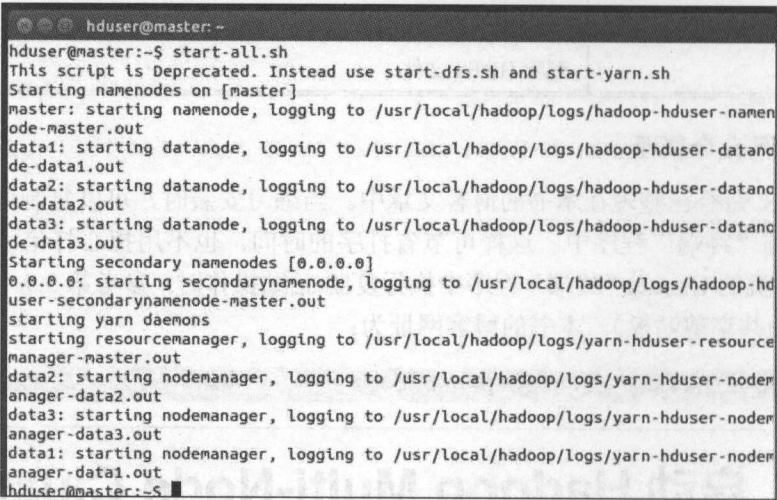


图 6-3 运行 start-all.sh 命令来启动 Hadoop Multi-Node Cluster

6.2 创建与查看 HDFS 目录

步骤 01 创建 HDFS 目录

在“终端”程序中输入下列命令，创建 HDFS 目录：

➤ 创建 user 目录

```
hadoop fs -mkdir /user
```

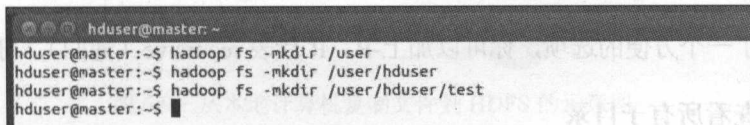
➤ 在 user 目录下，创建 hduser 子目录

```
hadoop fs -mkdir /user/hduser
```

➤ 在 hduser 目录下，创建 test 子目录

```
hadoop fs -mkdir /user/hduser/test
```

运行后屏幕显示界面如图 6-4 所示。



```
hduser@master:~$ hadoop fs -mkdir /user
hduser@master:~$ hadoop fs -mkdir /user/hduser
hduser@master:~$ hadoop fs -mkdir /user/hduser/test
hduser@master:~$
```

图 6-4 创建 HDFS 目录

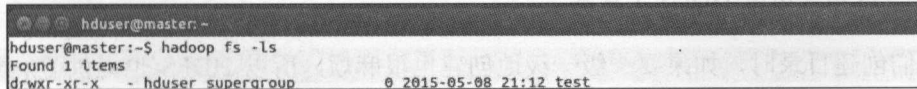
步骤 02 查看用户 HDFS 目录

在“终端”程序中输入下列命令：

➤ 查看之前创建的 HDFS 目录

```
hadoop fs -ls
```

因为当前登录的用户是 hduser，所以会显示/user/hduser 下的目录，也就是 test 目录，如图 6-5 所示。



```
hduser@master:~$ hadoop fs -ls
Found 1 items
drwxr-xr-x - hduser supergroup 0 2015-05-08 21:12 test
```

图 6-5 查看用户 HDFS 目录

步骤 03 查看 HDFS 完整目录

在“终端”程序中输入下列命令，查看之前创建的完整 HDFS 目录。因为 `hadoop fs -ls` 只能查看一级目录，所以必须一级一级目录地查看，如下列范例：

➤ 查看 HDFS 根目录

```
hadoop fs -ls /
```

➤ 查看 HDFS 的/user 目录

```
hadoop fs -ls /user
```

➤ 查看 HDFS 的/user/hduser 目录

```
hadoop fs -ls /user/hduser
```

运行后屏幕显示界面如图 6-6 所示。

```
hduser@master:~$ hadoop fs -ls /
Found 1 items
drwxr-xr-x - hduser supergroup          0 2015-05-08 09:51 /user
hduser@master:~$ hadoop fs -ls /user
Found 1 items
drwxr-xr-x - hduser supergroup          0 2015-05-08 09:51 /user/hduser
hduser@master:~$ hadoop fs -ls /user/hduser
Found 1 items
drwxr-xr-x - hduser supergroup          0 2015-05-08 09:51 /user/hduser/test
hduser@master:~$
```

图 6-6 查看 HDFS 完整目录

步骤 04 查看所有 HDFS 子目录

HDFS 提供了一个方便的选项，你可以加上-R，R 代表 recursive（递归），可进行如下操作：

➤ 一次查看所有子目录

```
hadoop fs -ls -R /
```

运行后屏幕显示界面如图 6-7 所示。

```
hduser@master:~$ hadoop fs -ls -R /
drwxr-xr-x          0 2015-05-08 09:51 /user
drwxr-xr-x          0 2015-05-08 09:51 /user/hduser
drwxr-xr-x          0 2015-05-08 09:51 /user/hduser/test
hduser@master:~$
```

图 6-7 查看所有 HDFS 子目录

步骤 05 一次创建所有 HDFS 子目录

当我们创建目录时，如果要一级一级地创建也很麻烦，所以 HDFS 也提供了-p 选项，可以帮助用户一次创建多级目录，在“终端”程序中输入下列命令：

➤ 创建多级 HDFS 目录

```
hadoop fs -mkdir -p /dir1/dir2/dir3
```

➤ 查看所有 HDFS 子目录

```
hadoop fs -ls -R /
```

执行后屏幕显示界面如图 6-8 所示。



图 6-8 一次创建所有 HDFS 子目录

6.3 从本地计算机复制文件到 HDFS

从本地计算机复制文件到 HDFS 的示意图如图 6-9 所示。

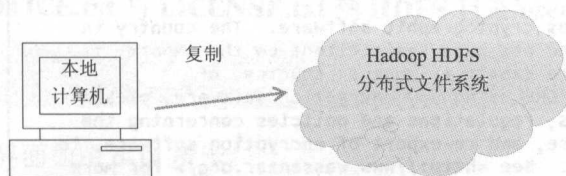


图 6-9 从本地计算机复制文件到 HDFS 的示意图

步骤 01 复制本地 (local) 文件到 HDFS

使用下列命令复制本地 (local) 文件到 HDFS:

➤ 复制本地文件到 HDFS 的目录

```
hadoop fs -copyFromLocal /usr/local/hadoop/README.txt /user/hduser/test
```

➤ 复制本地文件到 HDFS 的目录的 test1.txt

```
hadoop fs -copyFromLocal /usr/local/hadoop/README.txt /user/hduser/test/test1.txt
```

➤ 列出 HDFS 目录下的文件

```
hadoop fs -ls /user/hduser/test
```

运行后, 屏幕显示界面如图 6-10 所示。

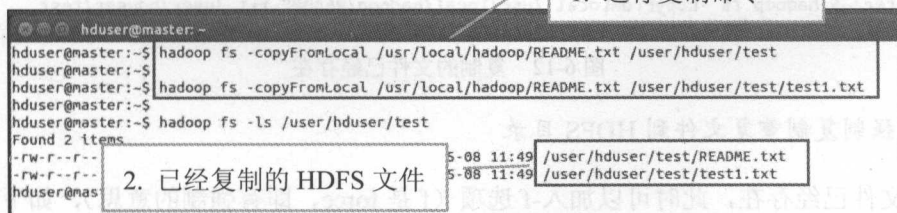


图 6-10 复制本地 (local) 文件到 HDFS

步骤 02 列出 HDFS 目录下的文件

我们可以在“终端”程序中输入下列命令：

➤ 列出 HDFS 目录下的文件内容

```
hadoop fs -cat /user/hduser/test/README.txt
```

运行后，屏幕显示界面如图 6-11 所示。

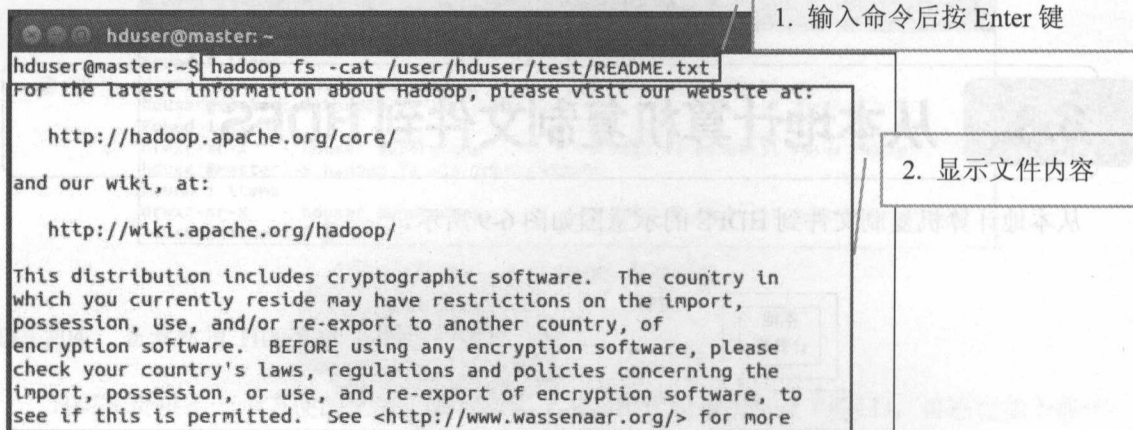


图 6-11 列出 HDFS 目录下的文件

不过，上列命令会一次列出所有文件内容，如果文件太大可以加上“|more”，就可以一页一页地显示，如下命令：

```
hadoop fs -cat /user/hduser/test/README.txt|more
```

步骤 03 复制重复文件到 HDFS 目录

当我们复制本地文件至 HDFS 目录时，如果文件已经存在，系统会回复：File exists，即文件已经存在，将不会复制。

➤ 复制本地文件到 HDFS 的目录时，文件已经存在

```
hadoop fs -copyFromLocal /usr/local/hadoop/README.txt /user/hduser/test
```

运行后屏幕显示界面如图 6-12 所示。

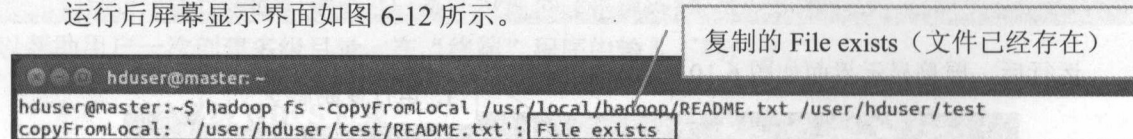


图 6-12 复制的文件已经存在

步骤 04 强制复制重复文件到 HDFS 目录

如果文件已经存在，此时可以加入-f 选项（f 是 force，即有强制的意思），如下列命令：

➤ 强制复制文件

```
hadoop fs -copyFromLocal -f /usr/local/hadoop/README.txt /user/hduser/test
```

强制复制本地文件/usr/local/hadoop/README.txt 到 HDFS 目录/user/hduser/test。
运行后屏幕显示界面如图 6-13 所示。

```
hduser@master: ~
hduser@master:~$ hadoop fs -copyFromLocal -f /usr/local/hadoop/README.txt /user/hduser/test
hduser@master:~$
```

图 6-13 强制复制重复文件至 HDFS 目录

步骤 05 复制多个本地文件到 HDFS 目录

也可以一次复制多个本地文件至 HDFS 目录，如下列命令：

➤ 同时复制 NOTICE.txt 与 LICENSE.txt 到 HDFS 目录/user/hduser/test

```
hadoop fs -copyFromLocal /usr/local/hadoop/NOTICE.txt
/usr/local/hadoop/LICENSE.txt /user/hduser/test
```

运行后的屏幕显示界面如图 6-14 所示。

```
hduser@master: ~
hduser@master:~$ hadoop fs -copyFromLocal /usr/local/hadoop/NOTICE.txt /user/hduser/test
hduser@master:~$ hadoop fs -ls /user/hduser/test
Found 5 items
-rw-r--r-- 3 hduser supergroup 15429 2015-05-08 12:02 /user/hduser/test/LICENSE.txt
-rw-r--r-- 3 hduser supergroup 101 2015-05-08 12:02 /user/hduser/test/NOTICE.txt
-rw-r--r-- 3 hduser supergroup 1366 2015-05-08 11:51 /user/hduser/test/README.txt
-rw-r--r-- 3 hduser supergroup 1366 2015-05-08 11:49 /user/hduser/test/test1.txt
```

图 6-14 复制多个本地文件到 HDFS 目录

步骤 06 copyFromLocal 复制目录到 HDFS 目录

也可以从本地计算机复制整个目录到 HDFS 目录，如下列命令：

➤ 复制整个本地计算机的目录/usr/local/hadoop/etc 到 HDFS 目录/user/hduser/test

```
hadoop fs -copyFromLocal /usr/local/hadoop/etc /user/hduser/test
```

➤ 列出在 HDFS 目录下的文件

```
hadoop fs -ls /user/hduser/test
```

运行界面如图 6-15 所示。

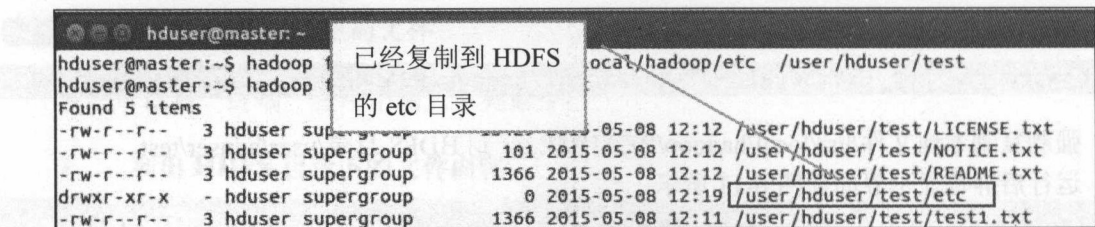


图 6-15 copyFromLocal 复制目录到 HDFS 目录

步骤 07 查看目录下所有的文件

之前的命令只是看到 etc 目录名称，还可以使用下列命令：

➤ 列出 HDFS 目录/user/hduser/test/etc 下的所有文件

```
hadoop fs -ls -R /user/hduser/test/etc
```

以上命令加上-R 选项（-R 是 Recursive），可列出 HDFS 目录下的所有文件，包含子目录：运行结果的屏幕显示界面如图 6-16 所示。

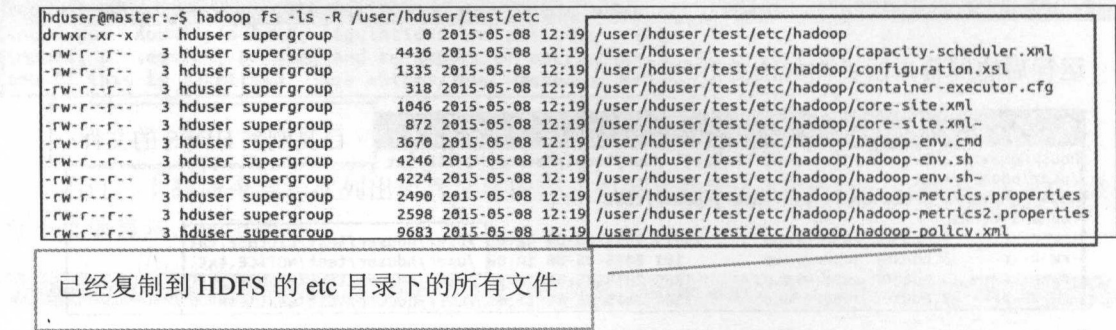


图 6-16 查看目录下所有的文件

步骤 08 使用 put 复制文件到 HDFS 目录

另外，也可以使用“-put”选项，复制文件到 HDFS 目录。使用“-put”与“-copyFromLocal”的不同之处是：如果文件已经存在，系统不会显示文件已经存在，而会直接覆盖，例如下列命令：

➤ 使用 put 复制文件到 HDFS 目录，会直接覆盖文件

```
hadoop fs -put /usr/local/hadoop/README.txt /user/hduser/test/test2.txt
```

运行后屏幕显示界面如图 6-17 所示。

```

hduser@master:~$ hadoop fs -put /usr/local/hadoop/README.txt /user/hduser/test/test2.txt
hduser@master:~$ hadoop fs -ls /user/hduser/test
Found 6 items
-rw-r--r-- 12:12 /user/hduser/test/LICENSE.txt
-rw-r--r-- 12:12 /user/hduser/test/NOTICE.txt
-rw-r--r-- 12:12 /user/hduser/test/README.txt
drwxr-xr-x 12:19 /user/hduser/test/etc
-rw-r--r-- 3 hduser supergroup 1366 2015-05-08 12:11 /user/hduser/test/test1.txt
-rw-r--r-- 3 hduser supergroup 1366 2015-05-08 12:28 /user/hduser/test/test2.txt
hduser@master:~$

```

已经复制至 HDFS 的 test2.txt 文件

图 6-17 使用 put 复制文件到 HDFS 目录

步骤 09 使用 put 命令接受 stdin (标准输入)

另外一个使用 `-put` 与 `-copyFromLocal` 的不同处是：“`-put`”可以接受 `stdin` (标准输入)，例如下列命令：

➤ 将原本显示在屏幕上的内容存储到 HDFS 文件

```
echo abc | hadoop fs -put - /user/hduser/test/echoin.txt
```

`echo abc` 原本是要显示在屏幕上的内容，现在通过“`|`”(即 `pipe` 管道)符号，传递给 `hadoop` 的命令，并且存储到 HDFS 目录下的文件 `echoin.txt` 中。

➤ 显示在 HDFS 的文件 echoin.txt 的内容

```
hadoop fs -cat /user/hduser/test/echoin.txt
```

运行后列出 `abc`，如图 6-18 所示。

```

hduser@master:~$ echo abc | hadoop fs -put - /user/hduser/test/echoin.txt
hduser@master:~$ hadoop fs -cat /user/hduser/test/echoin.txt
abc

```

echoin.txt 的文件内容

图 6-18 使用 put 命令接受 stdin (标准输入)

步骤 10 使用 put 命令将本地目录的列表存储到 HDFS 文件

另外一个范例是我们可以将：

➤ 本地目录的列表，存储到 HDFS 文件

```
ls /usr/local/hadoop | hadoop fs -put - /user/hduser/test/hadooplist.txt
```

原本 `ls /usr/local/hadoop` 命令会把本地目录的列表列在屏幕上，但是通过“`|`”符号(即 `pipe` 管道)传递给 `Hadoop` 命令，最后会存储到 HDFS 目录下的文件 `hadooplist.txt` 中。

➤ 显示 HDFS 的文件 hadooplist.txt 内容

```
hadoop fs -cat /user/hduser/test/hadooplist.txt
```

运行后屏幕显示界面如图 6-19 所示。

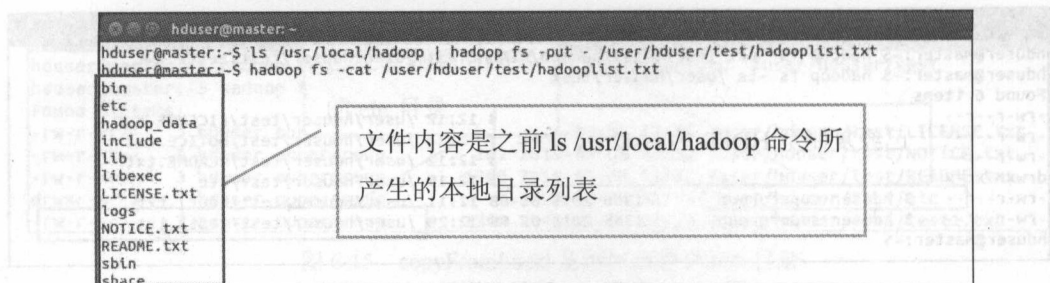


图 6-19 使用 put 命令将本地目录的列表存储到 HDFS 文件

6.4 将 HDFS 上的文件复制到本地计算机

将 HDFS 上的文件复制到本地计算机的示意图，如图 6-20 所示。



图 6-20 将 HDFS 上的文件复制到本地计算机的示意图

步骤 01 将 HDFS 上的文件复制到本地计算机 (local)

在“终端”程序中输入下列命令，将 HDFS 上的文件复制到本地。

➤ 在本地计算机创建 test 测试目录

```
mkdir test
```

➤ 切换到 test 目录

```
cd test
```

➤ 将 HDFS 的文件复制到本地计算机

```
hadoop fs -copyToLocal /user/hduser/test/hadooplist.txt
```

➤ 查看本地目录

```
ll
```

运行后的屏幕显示界面如图 6-21 所示。

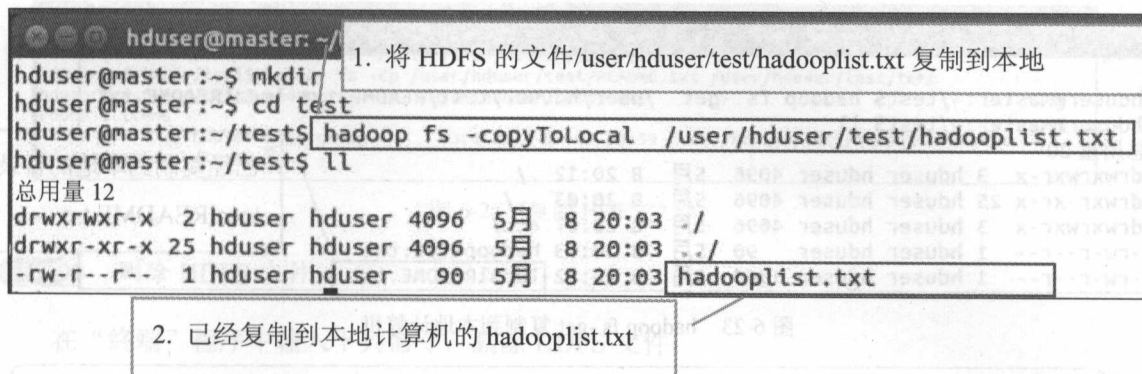


图 6-21 将 HDFS 上的文件复制到本地计算机

步骤 02 将 HDFS 上的目录复制到本地计算机

也可以使用下列命令：

➤ 将整个 HDFS 上的目录复制到本地计算机

```
hadoop fs -copyToLocal /user/hduser/test/etc
```

执行后屏幕显示界面如图 6-22 所示：

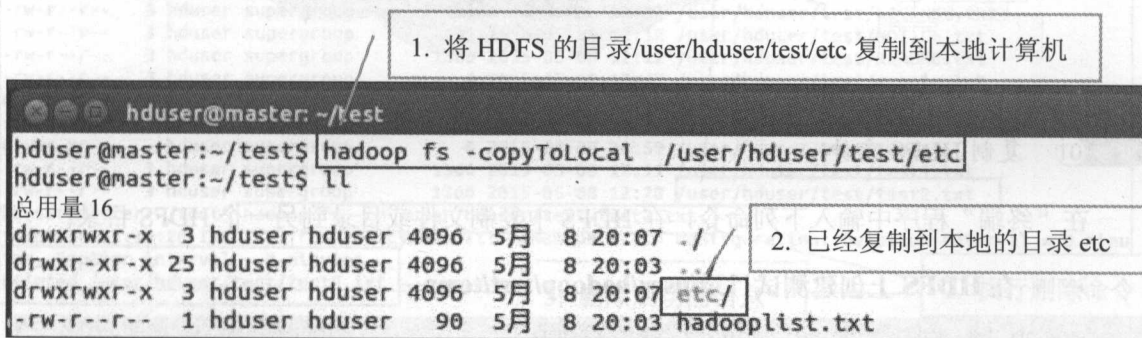


图 6-22 将 HDFS 上的目录复制到本地计算机

步骤 03 hadoop fs -get 复制到本地计算机

也可以使用下列“-get”命令：

➤ 将 HDFS 上的文件复制到本地计算机

```
hadoop fs -get /user/hduser/test/README.txt localREADME.txt
```

运行后屏幕显示界面如图 6-23 所示。

```
hduser@master: ~/test

hduser@master:~/test$ hadoop fs -get /user/hduser/test/README.txt localREADME.txt
hduser@master:~/test$ ll
总用量 20
drwxrwxr-x 3 hduser hduser 4096 5月 8 20:12 ./
drwxr-xr-x 25 hduser hduser 4096 5月 8 20:03 ../
drwxrwxr-x 3 hduser hduser 4096 5月 8 20:07 etc/
-rw-r--r-- 1 hduser hduser 90 5月 8 20:03 hadooplist.txt
-rw-r--r-- 1 hduser hduser 1366 5月 8 20:12 localREADME.txt
```

已经复制到本地的目录
localREADME.txt

图 6-23 hadoop fs -get 复制到本地计算机

6.5 复制与删除 HDFS 文件

如图 6-24 所示复制 HDFS 文件,指的是在 HDFS 中复制文件或目录到另一个 HDFS 目录。

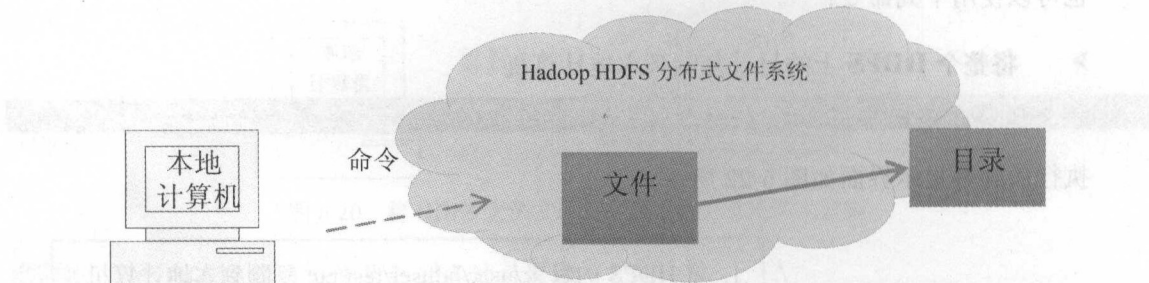


图 6-24 复制与删除 HDFS 文件

步骤 01 复制 HDFS 文件

在“终端”程序中输入下列命令,在 HDFS 中复制文件或目录到另一个 HDFS 目录:

- 在 HDFS 上创建测试目录 `/user/hadoop/test/temp`

```
hadoop fs -mkdir /user/hduser/test/temp
```

- 复制 HDFS 文件到 HDFS 测试目录

```
hadoop fs -cp /user/hduser/test/README.txt /user/hduser /test/temp
```

复制 HDFS 文件 `/user/hduser/test/README.txt`, 到 HDFS 测试目录 `/user/hadoop/test/temp`。

- 查看 HDFS 测试目录

```
hadoop fs -ls /user/hduser/test/temp
```

运行后屏幕显示界面如图 6-25 所示。

```

hduser@master: ~/test
hduser@master:~/test$ hadoop fs -mkdir /user/hduser/test/temp
hduser@master:~/test$ hadoop fs -cp /user/hduser/test/README.txt /user/hduser/test/temp
hduser@master:~/test$ hadoop fs -ls /user/hduser/test/temp
Found 1 items
-rw-r--r-- 3 hduser supergroup 1366 2015-05-08 20:59 /user/hduser/test/temp/README.txt
hduser@master:~/test$

```

图 6-25 复制 HDFS 文件

步骤 02 删除 HDFS 文件

在“终端”程序中输入下列命令，删除 HDFS 文件：

➤ 先查看准备要被删除的文件

```
hadoop fs -ls /user/hduser/test
```

➤ 删除 HDFS 文件

```
hadoop fs -rm /user/hduser/test/test2.txt
```

运行后屏幕显示界面如图 6-26 所示。

```

hduser@master: ~/test
hduser@master:~/test$ hadoop fs -ls /user/hduser/test
Found 10 items
-rw-r--r-- 3 hduser supergroup 8 2015-05-08 12:37 /user/hduser/test/-
-rw-r--r-- 3 hduser supergroup 15429 2015-05-08 12:12 /user/hduser/test/LICENSE.txt
-rw-r--r-- 3 hduser supergroup 101 2015-05-08 12:12 /user/hduser/test/NOTICE.txt
-rw-r--r-- 3 hduser supergroup 1366 2015-05-08 12:12 /user/hduser/test/README.txt
-rw-r--r-- 3 hduser supergroup 4 2015-05-08 12:47 /user/hduser/test/echoin.txt
drwxr-xr-x - hduser supergroup 0 2015-05-08 12:19 /user/hduser/test/etc
-rw-r--r-- 3 hduser supergroup 90 2015-05-08 12:53 /user/hduser/test/hadooplist.txt
drwxr-xr-x - hduser supergroup 0 2015-05-08 20:59 /user/hduser/test/temp
-rw-r--r-- 3 hduser supergroup 1366 2015-05-08 12:11 /user/hduser/test/test1.txt
-rw-r--r-- 3 hduser supergroup 1366 2015-05-08 12:28 /user/hduser/test/test2.txt
hduser@master:~/test$ hadoop fs -rm /user/hduser/test/test2.txt
15/05/08 21:10:26 INFO fs.TrashPolicyDefault: Namenode trash configuration: Deletion interval = 0 minu
tes, Empty interval = 0 minutes.
Deleted /user/hduser/test/test2.txt

```

1. 要删除的文件

2. 执行删除命令

3. 删除成功的信息

图 6-26 删除 HDFS 文件

步骤 03 删除 HDFS 目录

在“终端”程序中输入下列命令，删除 HDFS 目录：

➤ 查看准备要被删除的 HDFS 目录

```
hadoop fs -ls /user/hduser/test
```

➤ 删除 HDFS 目录

```
hadoop fs -rm -R /user/hduser/test/etc
```

运行后屏幕显示界面如图 6-27 所示。

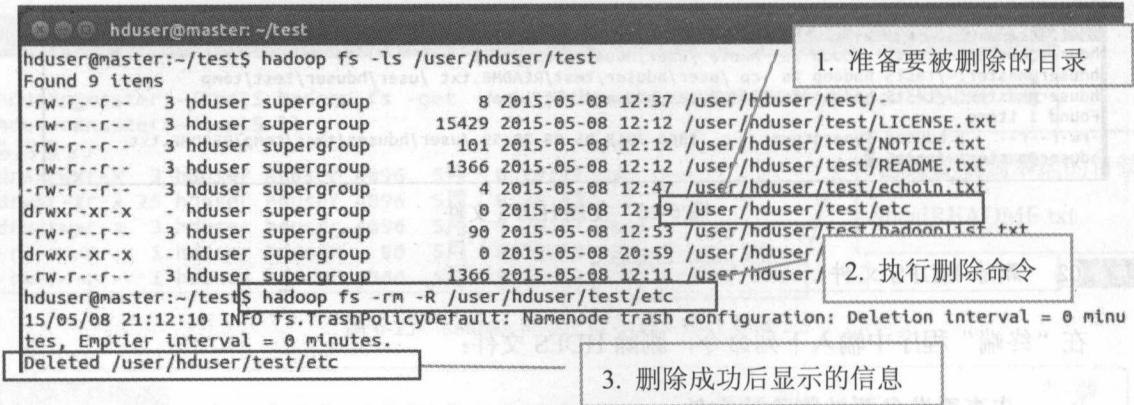


图 6-27 删除 HDFS 目录

6.6 在 Hadoop HDFS Web 用户界面浏览 HDFS

之前我们使用了 HDFS 命令来查看 HDFS 目录。Hadoop 另外还提供了更方便的功能，可以在 Hadoop HDFS Web 用户界面来浏览 HDFS 目录或文件，如图 6-28 所示。

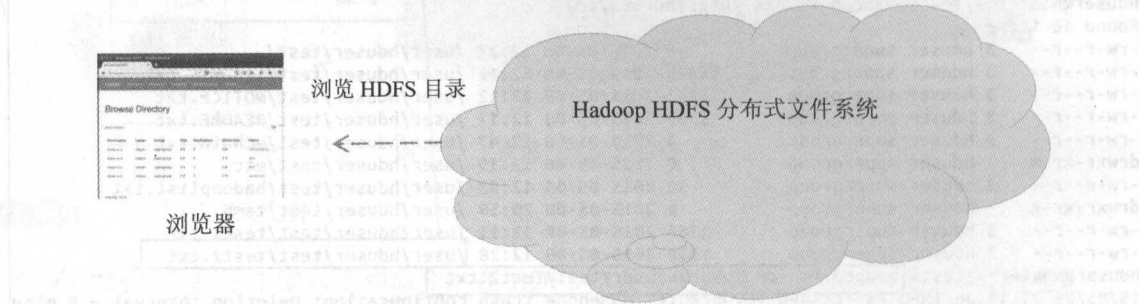


图 6-28 通过 Hadoop HDFS Web 用户界面浏览 HDFS 的示意图

步骤 01 浏览 Hadoop HDFS Web 界面查看 HDFS

网址	说明
http://master:50070	Hadoop HDFS Web 界面网址

参照下列步骤，如图 6-29 所示，就可以看到我们之前创建的 HDFS 目录与文件：

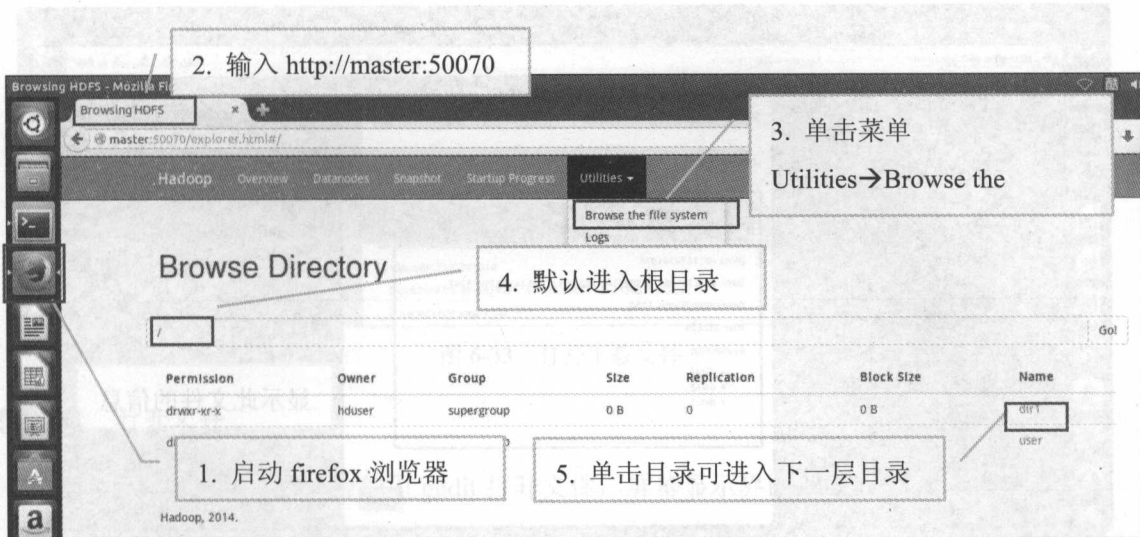


图 6-29 通过 Hadoop HDFS Web 用户界面浏览 HDFS

步骤 02 输入完整的目录路径来查看文件

也可以直接输入完整的目录路径来查看文件，如图 6-30 所示。

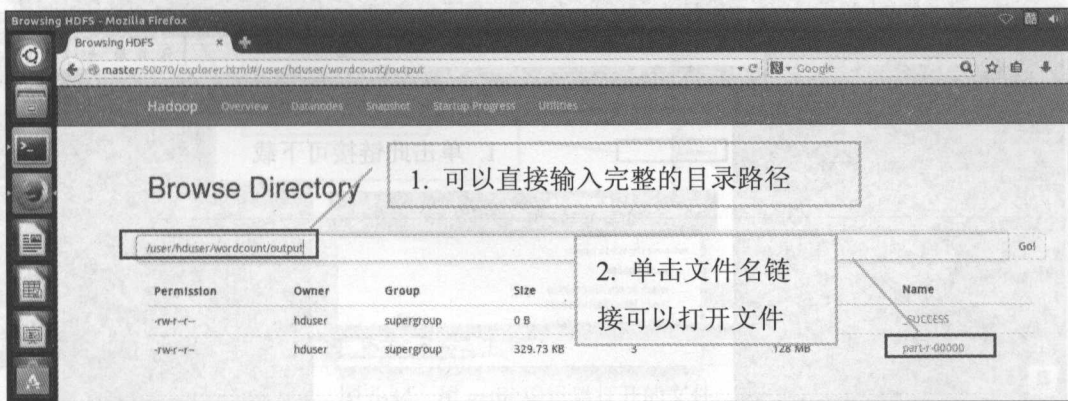


图 6-30 输入完整的目录路径来查看文件

步骤 03 查看 HDFS 文件的完整信息

单击文件名链接后，会打开 File Information 对话框显示此文件的信息，如图 6-31 所示。

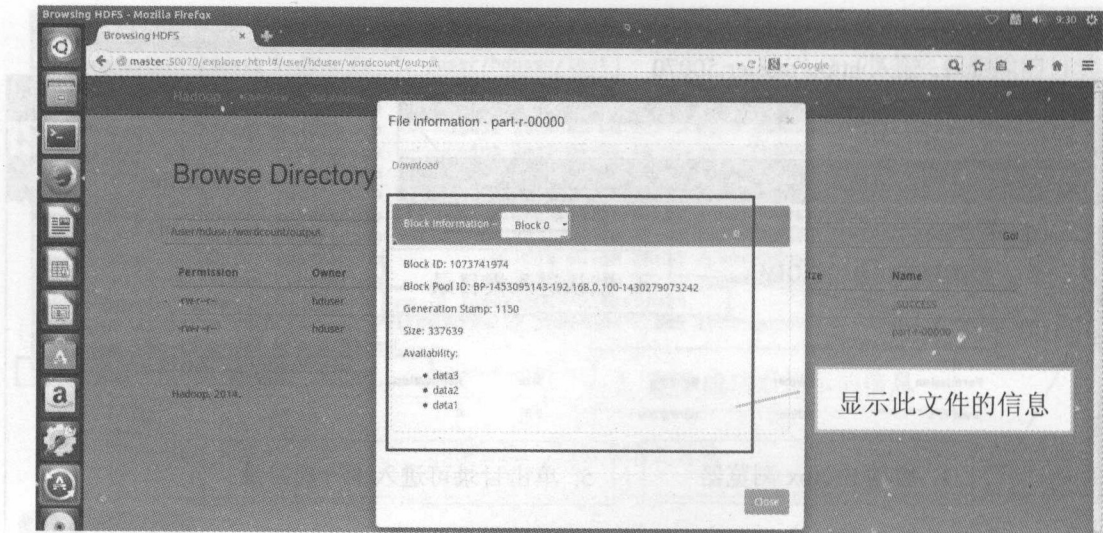


图 6-31 查看 HDFS 文件的完整信息

步骤 04 下载 HDFS 文件

可以参照下列步骤来下载 HDFS 文件，如图 6-32 所示。

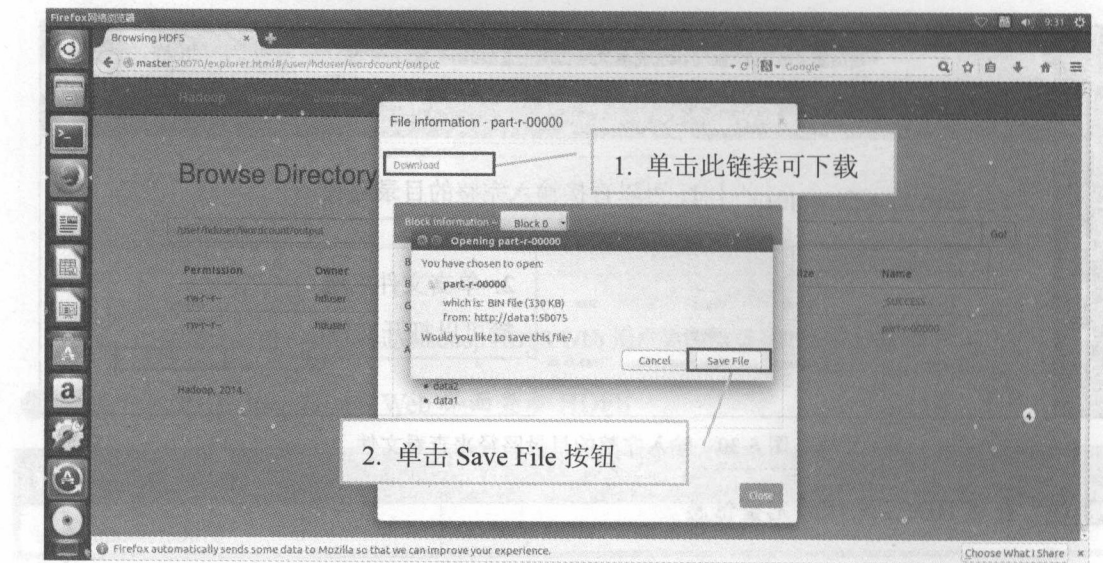


图 6-32 下载 HDFS 文件

步骤 05 打开下载文件

下载文件后可以查看下载的文件，如图 6-33 所示。

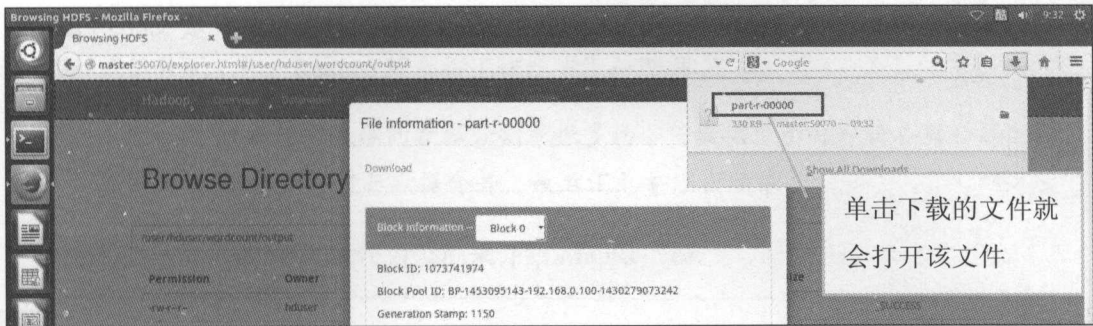


图 6-33 打开下载文件

步骤 06 打开文件

因为是文本文件，默认会使用 `gedit` 打开文件，屏幕显示界面如图 6-34 所示。

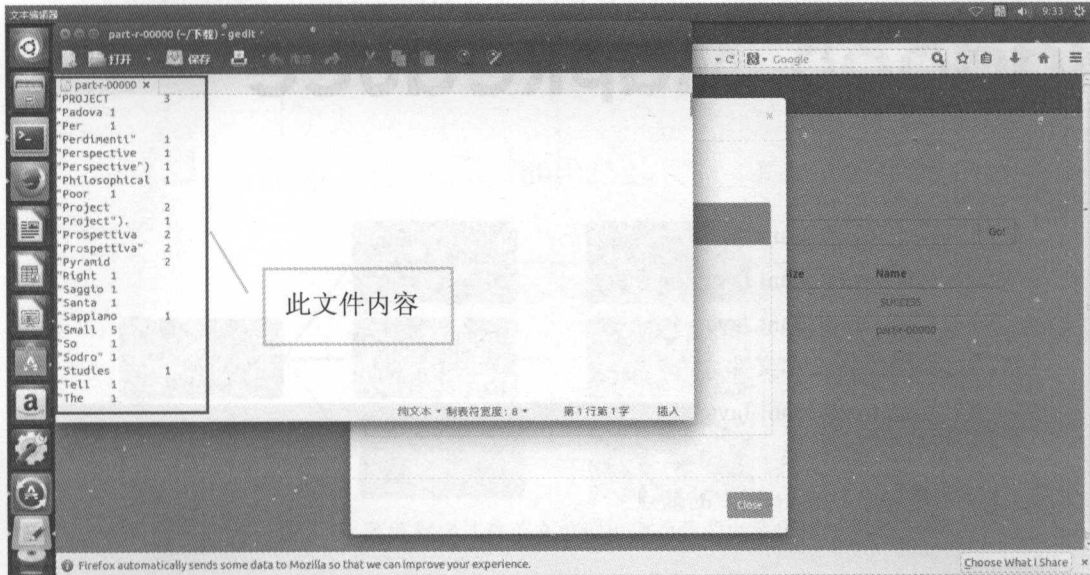


图 6-34 用 gedit 编辑器打开的文件

第 7 章

Hadoop MapReduce

- 7.1 介绍 wordCount.Java
- 7.2 编辑 wordCount.Java
- 7.3 编译 wordCount.Java
- 7.4 创建测试文本文件
- 7.5 运行 wordCount.Java
- 7.6 查看运行结果
- 7.7 Hadoop MapReduce 的缺点

MapReduce 是一种程序开发模式，可以使用大量服务器来并行处理。MapReduce，简单地说，Map 就是分配工作、Reduce 就是将工作结果汇总整理。

- 首先使用 Map 将待处理的数据分割成很多的小份数据，由每台服务器分别运行。
- 再通过 Reduce 程序进行数据合并，最后汇总整理出结果。

本章将以 wordCountt.Java 作为范例来介绍 MapReduce。

7.1 介绍 wordCount.Java

下列为 word count 范例，要计算文件中每一个英文单词出现的次数，步骤如图 7-1 所示。

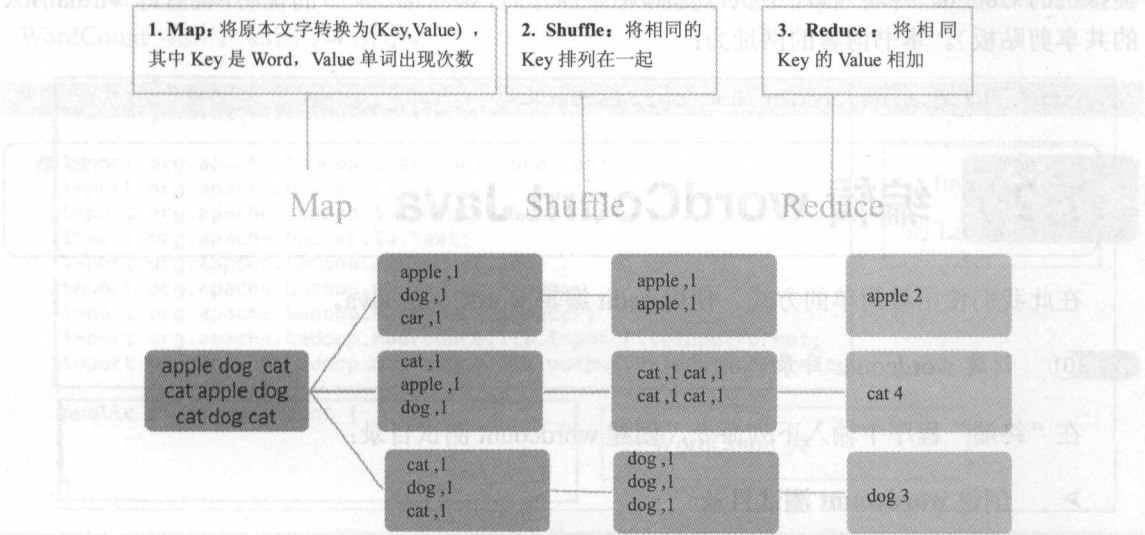


图 7-1 范例程序计算英文单词出现次数的大致步骤

在 Hadoop 说明文件中就具有 wordcount.Java 的程序代码，在浏览器输入下列网址：

```
http://hadoop.apache.org/docs/current/hadoop-MapReduce-client/hadoop-MapReduce-client-core/MapReduceTutorial.html
```

就可以看到详细说明。本章将以 wordcount.Java 示范 Map Reduce 的运行方式。

➤ WordCount 开发步骤如下

本章所有命令，都在 master 虚拟机的“终端”程序中运行。

顺序	开发步骤	说明
1	编辑 wordCount.Java	先使用 gedit 编辑 WordCount.Java
2	编译 wordCount.Java	编译、打包 wordCount.Java 程序

(续表)

顺序	开发步骤	说明
3	创建测试文本文件	使用 Hadoop 目录下的 LICENSE.txt 文件作为测试文件，并上传文本文件至 HDFS
4	运行 wordCount.Java	在 Hadoop 环境运行 WordCount
5	查看运行结果	运行会产生输出文件并存储到 HDFS 中，可使用 HDFS 命令查看

➤ 本章命令整理

以下命令已整理在本书的博客文章中。当练习安装时，可以复制博客文章中的命令，然后粘贴到“终端”程序中。这样可节省打字的时间，也不用担心打错字（如果无法在 VirtualBox 虚拟机的 Ubuntu“终端”程序中执行复制/粘贴操作时，参考第 3.9 节的说明，设置好 VirtualBox 的共享剪贴板）。本书博客的网址为：

<http://blog.sina.com.cn/hadoopsparkbook>

7.2 编辑 wordCount.Java

在此我们使用最简单的方式，使用 gedit 编辑 wordCount.Java。

步骤 01 创建 wordcount 目录

在“终端”程序中输入下列命令，创建 wordcount 测试目录：

➤ 创建 wordcount 测试目录

```
mkdir -p ~/wordcount/input
```

➤ 切换至 wordcount 测试目录

```
cd ~/wordcount
```

运行后屏幕显示界面如图 7-2 所示。

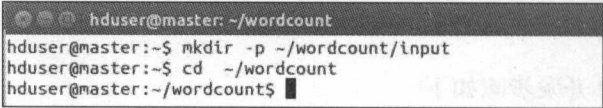


图 7-2 创建 wordcount 目录

步骤 02 编辑 WordCount.Java

在“终端”程序中输入下列命令：

➤ 使用 gedit 编辑 WordCount.Java

```
sudo gedit WordCount.Java
```

按 Enter 键之后，就会打开 WordCount.Java 空白文件，如图 7-3 所示。



图 7-3 编辑 WordCount.Java

步骤 03 Import 相关的 Lib 并创建 WordCount 类

首先要 Import 相关的 Lib 链接库，并且创建 WordCount 类 class。后续的程序都是引入到 WordCount 类中，如图 7-4 所示。

```
import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {
```

1. Import 相关的 Lib 链接库

2. WordCount 类

图 7-4 Import 相关的 Lib 并创建 WordCount 类

步骤 04 创建 main function

接下来创建 main function 程序。main function 是程序的起点，在 main function 中，主要设置 map 与 reduce 类，如图 7-5 所示。

```
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

1. 设置 TokenizerMapper 类

2. 设置 IntSumReducer 类

图 7-5 创建 main function

步骤 05 创建 TokenizerMapper 类

在 TokenizerMapper 类的 map 方法中, 先创建 StringTokenizer, 然后使用 while (itr.hasMoreTokens()) 读取每一个 word, 并写入 context.write(word, one), 创建 key/value (word, 1), 如图 7-6 所示。

```
public static class TokenizerMapper
    extends Mapper<Object, Text, Text, IntWritable>{

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context
        ) throws IOException, InterruptedException {
        StringTokenizer itr = new StringTokenizer(value.toString());
        while (itr.hasMoreTokens()) {
            word.set(itr.nextToken());
            context.write(word, one);
        }
    }
}
```

图 7-6 TokenizerMapper 类

步骤 06 创建 IntSumReducer 类

在 IntSumReducer 类的 reduce 方法中, 先使用 for (IntWritable val : values) 读取每一个数值, 并使用 sum += val.get() 求和, 最后写入结果, 如图 7-7 所示。

```
public static class IntSumReducer
    extends Reducer<Text, IntWritable, Text, IntWritable> {
    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values,
        Context context
        ) throws IOException, InterruptedException {

        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        result.set(sum);
        context.write(key, result);
    }
}
```

图 7-7 创建 IntSumReducer 类

步骤 07 编辑完成的 wordCount.Java

编辑完成后, 单击“保存”按钮、再单击“关闭”按钮, 如图 7-8 所示。

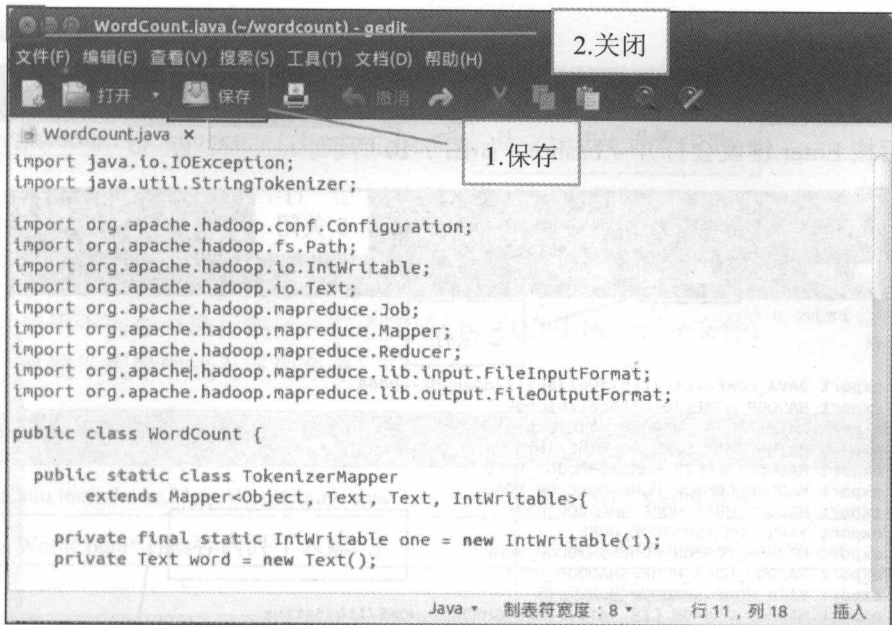


图 7-8 编辑完成的 wordCount.Java

步骤 08 查看编辑完成的 wordCount.Java

保存后，可以使用以下命令：

➤ 查看之前编辑完成的 wordCount.Java

11

运行后，屏幕显示界面如图 7-9 所示。

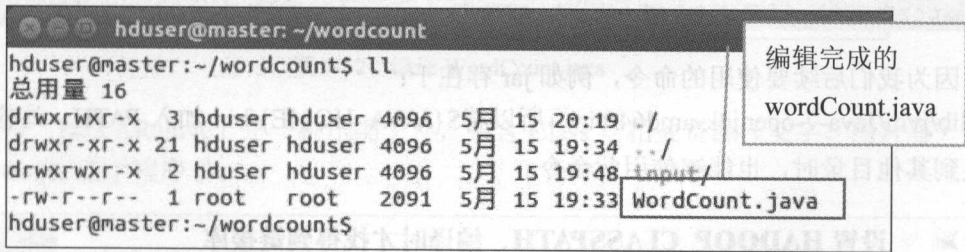


图 7-9 查看编辑完成的 wordCount.Java

7.3 编译 wordCount.Java

接下来要编译 wordCount.Java，但是需要先设置环境变量。

步骤 01 修改编译所需要的环境变量文件

在“终端”程序中输入下列命令：

➤ 编辑 ~/.bashrc

```
sudo gedit ~/.bashrc
```

输入后按 Enter 键就会打开 ~/.bashrc，如图 7-10 所示。

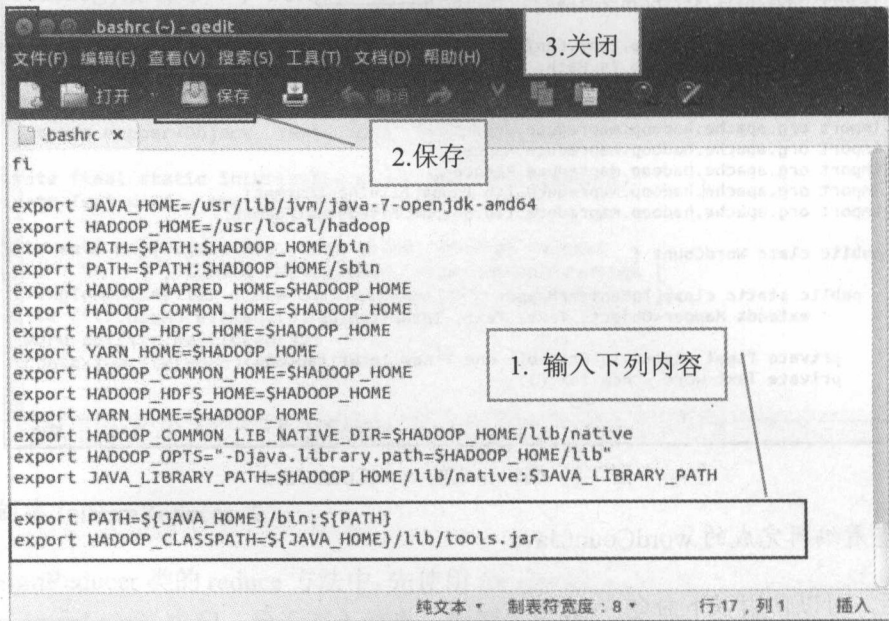


图 7-10 修改编译所需要的环境变量文件

上列命令说明如下：

➤ 设置 PATH

```
export PATH=${JAVA_HOME}/bin:${PATH}
```

因为我们后续要使用的命令，例如 jar 存在于：
/usr/lib/jvm/Java-7-openjdk-amd64/bin，所以将 \${JAVA_HOME}/bin 加入 PATH。这样，让我们切换到其他目录时，也能够使用此命令。

➤ 设置 HADOOP_CLASSPATH，编译时才找得到链接库

```
export HADOOP_CLASSPATH=${JAVA_HOME}/lib/tools.jar
```

步骤 02 让 ~/.bashrc 修改的设置值生效

当我们修改 ~/.bashrc 之后，可以先从系统注销再重新登录，该设置就会生效。或使用 source 命令让 ~/.bashrc 设置立即生效。

在“终端”程序中输入下列命令：

➤ 让 ~/.bashrc 设置生效

```
source ~/.bashrc
```

屏幕显示界面如图 7-11 所示。

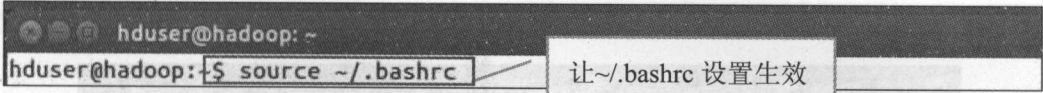


图 7-11 让 ~/.bashrc 修改的设置值生效

步骤 03 开始编译

你可以使用下列命令进行编译，编译完成后，就会生成 wc.jar 文件。
请在“终端”程序中输入下列命令：

命令	说明
hadoop com.sun.tools.Javac.Main WordCount.java	编译 WordCount 程序
jar cf wc.jar WordCount*.class	把 WordCount 类打包成 wc.jar
ll	查看目录内容

运行结果如图 7-12 所示。

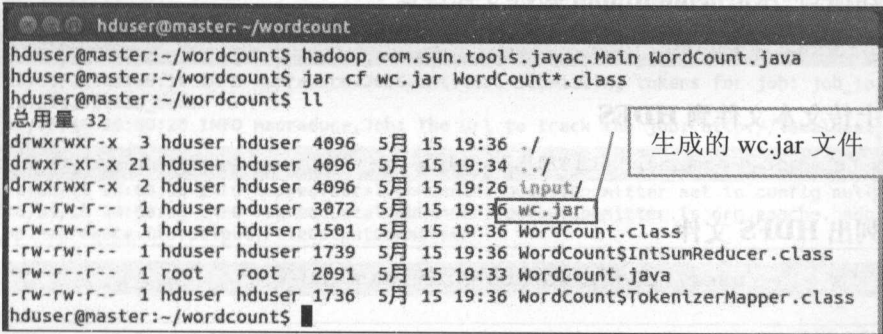


图 7-12 编译 WordCount.java

jar 文件（Java ARchive）是一种压缩文件，包含 Class 与相关资源（文字、图片等），之后就运行 wc.jar 这个程序了。

7.4 创建测试文本文件

为了测试 WordCount 程序，我们可以使用 Hadoop 目录下的 LICENSE.txt 文件。

步骤 01 复制 LICENSE.txt

在“终端”程序中输入下列命令，下载文本文件：

➤ 切换到输入目录

```
cp /usr/local/hadoop/LICENSE.txt ~/wordcount/input
```

```
ll ~/wordcount/input
```

运行结果如图 7-13 所示。

```
hduser@master: ~
hduser@master:~$ cp /usr/local/hadoop/LICENSE.txt ~/wordcount/input
hduser@master:~$ ll ~/wordcount/input
总用量 544
drwxrwxr-x 2 hduser hduser 4096 5月 15 19:48 ./
drwxrwxr-x 3 hduser hduser 4096 5月 15 19:36 ../
-rw-r--r-- 1 hduser hduser 15429 5月 15 19:48 LICENSE.txt
```

图 7-13 复制 LICENSE.txt

步骤 02 上传到 HDFS 目录

接下来，我们将之前下载的文本文件从本地复制到 HDFS。

在“终端”程序中输入下列命令：

➤ 在 HDFS 创建目录

```
hadoop fs -mkdir -p /user/hduser/wordcount/input
```

➤ 切换到~/wordcount/input 数据文件目录

```
cd ~/wordcount/input
```

➤ 上传文本文件到 HDFS

```
hadoop fs -copyFromLocal LICENSE.txt /user/hduser/wordcount/input
```

➤ 列出 HDFS 文件

```
hadoop fs -ls /user/hduser/wordcount/input
```

运行的结果如图 7-14 所示，已将 LICENSE.txt 上传到 HDFS 的/user/hduser/wordcount/input 目录。

```
hduser@master: ~/wordcount/input
hduser@master:~$ hadoop fs -mkdir -p /user/hduser/wordcount/input
hduser@master:~$ cd ~/wordcount/input
hduser@master:~/wordcount/input$ hadoop fs -copyFromLocal LICENSE.txt /user/hduser/wordcount/input
hduser@master:~/wordcount/input$ hadoop fs -ls /user/hduser/wordcount/input
Found 2 items
-rw-r--r-- 3 hduser supergroup 15429 2016-05-15 19:57 /user/hduser/wordcount/input/LICENSE.txt
```

图 7-14 将之前下载的文本文件上传到 HDFS 目录

7.5 运行 wordCount.Java

在“终端”程序中输入下列命令，运行 WordCount 程序：



➤ 切换目录

```
cd ~/wordcount
```

➤ 运行 WordCount 程序

```
hadoop jar wc.jar WordCount /user/hduser/wordcount/input/LICENSE.txt
/user/hduser/wordcount/output
```

运行 word count 程序，语句格式为“hadoop jar wc.jar [输入文件] [输出目录]”。如上述命令所示，程序会读取输入文件/user/hduser/wordcount/input/LICENSE.txt，处理完成后，将结果写入/user/hduser/wordcount/output 目录。

运行的结果如图 7-15 所示。

```
hduser@master: ~/wordcount
hduser@master:~$ cd ~/wordcount
hduser@master:~/wordcount$ hadoop jar wc.jar WordCount /user/hduser/wordcount/in
put/LICENSE.txt /user/hduser/wordcount/output
16/05/15 20:00:18 INFO Configuration.deprecation: session.id is deprecated. Inst
ead, use dfs.metrics.session-id
16/05/15 20:00:18 INFO jvm.JvmMetrics: Initializing JVM Metrics with processName
=JobTracker, sessionId=
16/05/15 20:00:19 WARN mapreduce.JobSubmitter: Hadoop command-line option parsin
g not performed. Implement the Tool interface and execute your application with
ToolRunner to remedy this.
16/05/15 20:00:19 INFO input.FileInputFormat: Total input paths to process : 1
16/05/15 20:00:19 INFO mapreduce.JobSubmitter: number of splits:1
16/05/15 20:00:19 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_lo
cal2102721529_0001
16/05/15 20:00:20 INFO mapreduce.Job: The url to track the job: http://localhost
:8080/
16/05/15 20:00:20 INFO mapreduce.Job: Running job: job_local2102721529_0001
16/05/15 20:00:20 INFO mapred.LocalJobRunner: OutputCommitter set in config null
16/05/15 20:00:20 INFO mapred.LocalJobRunner: OutputCommitter is org.apache.hado
op.mapreduce.lib.output.FileOutputCommitter
```

图 7-15 wordCount 程序的运行结果

7.6 查看运行结果

执行后，程序会将结果存储在/user/hduser/wordcount/output 目录中。参照下列步骤，查看结果。

步骤 01 查看输出目录

在“终端”程序中输入下列命令，查看运行结果。

➤ 查看 HDFS 的目录

```
hadoop fs -ls /user/hduser/wordcount/output
```

屏幕显示界面如图 7-16 所示，生成了两个文件。

- _SUCCESS: 代表程序运行成功。

- part-r-00000: 运行结果的文本文件。

```
hduser@master: ~/wordcount
hduser@master:~/wordcount$ hadoop fs -ls /user/hduser/wordcount/output
Found 2 items
-rw-r--r-- 3 hduser supergroup 0 2016-05-15 20:00 /user/hduser/wordcount/output/_SUCCESS
-rw-r--r-- 3 hduser supergroup 8006 2016-05-15 20:00 /user/hduser/wordcount/output/part-r-00000
hduser@master:~/wordcount$
```

图 7-16 查看 wordCount 程序的运行结果

步骤 02 查看输出文件

在“终端”程序中输入下列命令，查看输出文件：

➤ 查看 HDFS 中的输出文件内容

```
hadoop fs -cat /user/hduser/wordcount/output/part-r-00000|more
```

输出结果如图 7-17 所示，会列出每个英文单词出现的次数：

```
hduser@master: ~/wordcount
hduser@master:~/wordcount$ hadoop fs -cat /user/hduser/wordcount/output/part-r-00000|more
"AS" 4
"Contribution" 1
"Contributor" 1
"Derivative" 1
"Legal" 1
"License" 1
"License"); 1
"Licensor" 1
"NOTICE" 1
"Not" 1
"Object" 1
"Source" 1
"Work" 1
"You" 1
"Your") 1
"[]" 1
"control" 1
"printed" 1
"submitted" 1
(50%) 1
(C) 1
(Don't 1
(INCLUDING 2
```

图 7-17 查看 HDFS 中的输出文件内容

7.7 Hadoop MapReduce 的缺点

Hadoop MapReduce 有以下缺点：

- 程序设计模式不容易使用，而且 Hadoop 的 Map Reduce API 太过低级，很难提高开发者的效率。
- 有运行效率的问题，MapReduce 需要将中间产生的数据保存到硬盘中，因此会有读写数据延迟的问题。
- 不支持实时处理，它原始的设计就是以批处理为主。

这些缺点将在下一章介绍的 Spark 可以解决。

第 8 章

Spark的安装与介绍

- 8.1 Spark 的 Cluster 模式架构图
- 8.2 Scala 的介绍与安装
- 8.3 安装 Spark
- 8.4 启动 spark-shell 交互界面
- 8.5 设置 spark-shell 显示信息
- 8.6 启动 Hadoop
- 8.7 本地运行 spark-shell 程序
- 8.8 在 Hadoop YARN 运行 spark-shell
- 8.9 构建 Spark Standalone Cluster 执行环境
- 8.10 在 Spark Standalone 运行 spark-shell

8.1 Spark 的 Cluster 模式架构图

在 Spark 官网文件中，可以看到 Spark 的 Cluster 模式架构图（见图 8-1），在浏览器输入下列网址：

<http://spark.apache.org/docs/latest/cluster-overview.html>

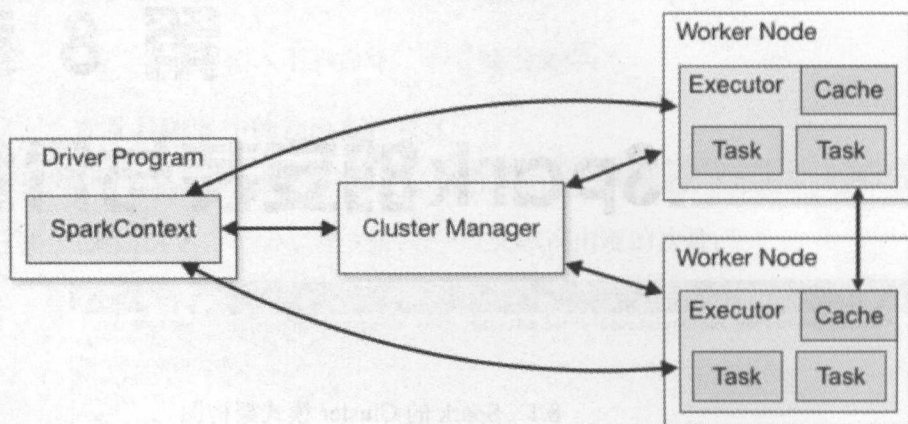


图 8-1 Spark 的 Cluster 模式架构图

参考图 8-1：

- DriverProgram 就是程序员所设计的 Spark 程序，在 Spark 程序中必须定义 SparkContext，它是开发 Spark 应用程序的入口。
- SparkContext 通过 Cluster Manager 管理整个集群，群集中包含多个 Worker Node，在每个 Worker Node 都有 Executor 负责执行任务。

SparkContext 通过 Cluster Manager 管理整个集群 Cluster 的好处是：所设计的 Spark 程序可以在不同的 Cluster 模式下运行。

➤ **Cluster Manager** 可以在下列模式下运行：

- 本地运行 (Local Machine) ——这只需要在程序中 import Spark 的链接库就可以实现。在本地运行时，对于只安装在一台计算机上时最方便，适合刚入门时学习、测试使用。
- Spark Standalone Cluster——这是由 Spark 提供的 Cluster 管理模式，如果没有架设 Hadoop Multi Node Cluster，则可以架设 Spark Standalone Cluster，实现多台计算机并行计算。在这个模式下仍然可以直接存取 Local Disk 或是 HDFS。
- Hadoop YARN (Yet Another Resource Negotiator)，它是 Hadoop 2.x 新架构中更高效率的资源管理核心。Spark 可以在 YARN 上运行，让 YARN 帮助它进行多台机器

的资源管理。

- 在云端运行——针对更大型规模的计算工作，本地机器或自建集群的计算能力恐怕难以满足。此时可以将 Spark 程序在云平台上运行，例如 AWS 的 EC2 平台。使用云计算的好处是不需要自己架设的费用，用多少付多少，随时可扩充。

本章将介绍如何在本地（Local Machine）、Spark Standalone Cluster、Hadoop YARN 运行 spark-shell。

➤ Spark 安装命令整理

以下命令已整理在本书的博客文章中。当练习安装时，可以复制博客文章中的命令，然后粘贴到“终端”程序中。这样可节省打字的时间，也不用担心打错字（如果无法在 VirtualBox 虚拟机的 Ubuntu“终端”程序中执行复制/粘贴操作时，参考第 3.9 节的说明，设置好 VirtualBox 的共享剪贴板）。本书博客的网址为：

<http://blog.sina.com.cn/hadoosparkbook>

8.2 Scala 的介绍与安装

Spark 支持 Scala、Java 和 Python 等语言，不过 Spark 本身是用 scala 语言开发的，所以在 Spark 应用程序开发中，Scala 被认为是当前和 Spark 兼容最好的语言。Scala 具有以下特点：

- Scala 可编译为 Java bytecode 字节码，也就是说它可以在 JVM（Java Virtual Machine）上运行，具备跨平台能力。
- 现有 Java 的链接库都可以使用，可以继续使用丰富的 Java 开放源码生态系统。
- Scala 是一种函数式语言。在函数式语言中，函数也是值，与整数字符串等处于同一地位。函数可以作为参数传递给其他函数。
- Scala 是一种纯面向对象的语言，所有的东西都是对象，而所有的操作都是方法。

因为 Spark 本身是以 Scala 开发的，所以必须先安装 Scala。我们将安装在 master 虚拟机上。

步骤 01 下载 Scala 网址

在这个 Scala 下载网页可以看到，各种不同版本的 Scala：

<http://www.scala-lang.org/files/archive/>

如图 8-2 所示，网页上有各种不同版本的 Scala。

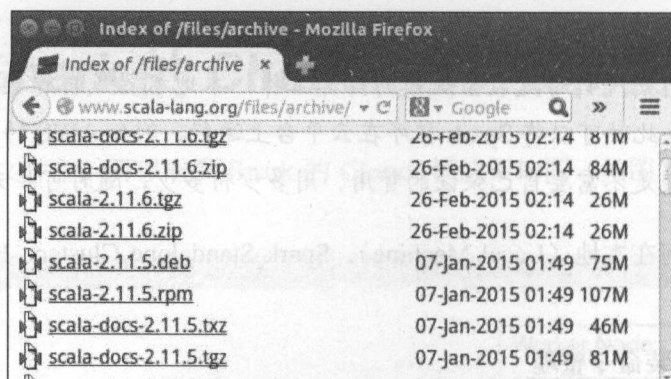


图 8-2 下载 Scala 的网站

步骤 02 下载 Scala

在本书中使用的是 scala-2.11.6 版本，在 master 虚拟机的“终端”程序中输入下列命令：

➤ 下载 scala-2.11.6 版本

```
wget http://www.scala-lang.org/files/archive/scala-2.11.6.tgz
```

运行后屏幕显示界面如图 8-3 所示。



图 8-3 下载 Scala

步骤 03 解压缩 Scala

在“终端”程序中输入下列命令：

➤ 解压缩 scala-2.11.6.tgz 到 scala-2.11.6 目录

```
tar xvf scala-2.11.6.tgz
```

运行后屏幕显示界面如图 8-4 所示。

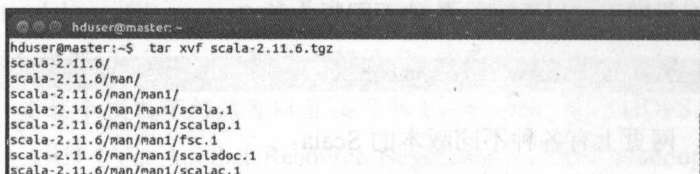


图 8-4 解压缩 Scala

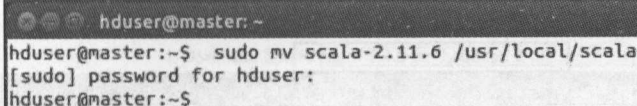
步骤 04 把 Scala 移运到/usr/local 目录

在“终端”程序中输入下令命令：

➤ 把 scala-2.11.6 移动到/usr/local 目录

```
sudo mv scala-2.11.6 /usr/local/scala
```

运行后屏幕显示界面如图 8-5 所示。



```
hduser@master:~$ sudo mv scala-2.11.6 /usr/local/scala
[sudo] password for hduser:
hduser@master:~$
```

图 8-5 把 Scala 移运到/usr/local 目录

步骤 05 Scala 用户环境变量的设置

在“终端”程序中输入下列命令：

➤ 启动 gedit 编辑 ~/.bashrc

```
sudo gedit ~/.bashrc
```

按下 Enter 键之后，会启动 gedit 编辑来 ~/.bashrc，输入如图 8-6 所示的内容。

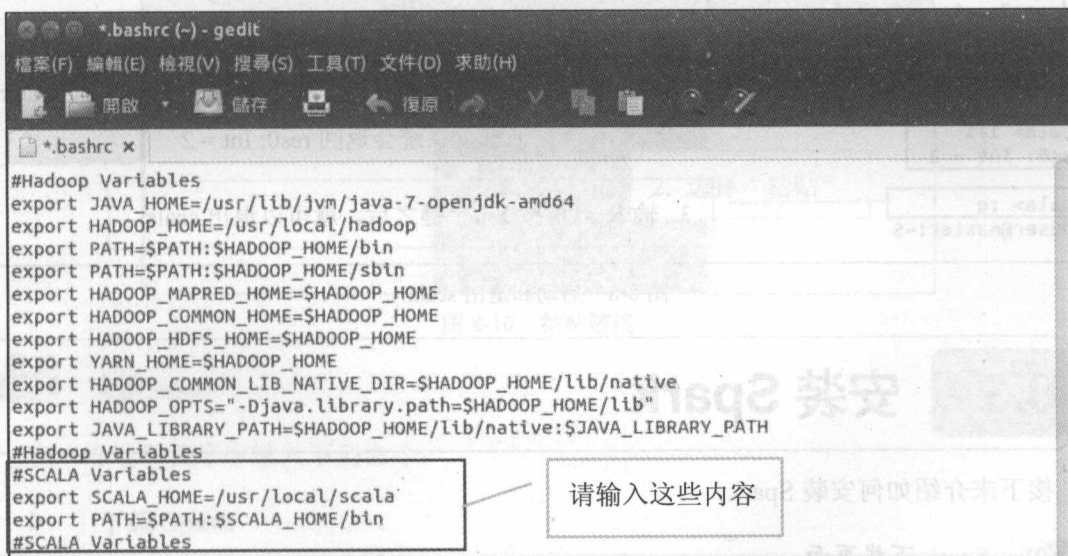


图 8-6 设置 Scala 用户的环境变量

说明如下：

➤ 设置 SCALA_HOME 也就是 scala 安装目录: /usr/local/scala

```
export SCALA_HOME=/usr/local/scala
```

➤ 设置 PATH 环境变量

```
export PATH=$PATH:$SCALA_HOME/bin
```

设置 PATH 环境变量加入\$SCALA_HOME/bin，让用户在不同的目录下都可以执行 scala 程序。

步骤 06 使 ~/.bashrc 修改生效

可以从系统注销再重新登录，或使用 source ~/.bashrc 命令让用户环境变量的设置生效，如图 8-7 所示。

```
source ~/.bashrc
```

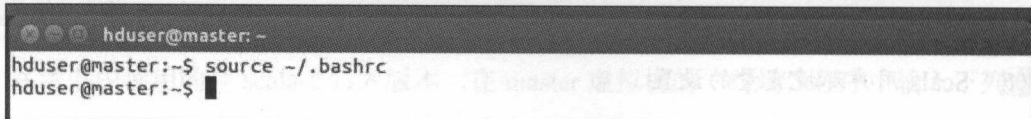


图 8-7 使 ~/.bashrc 修改生效

步骤 07 启动 scala

scala 至此已安装完成，接下来参照如图 8-8 的说明进入 scala 交互界面：

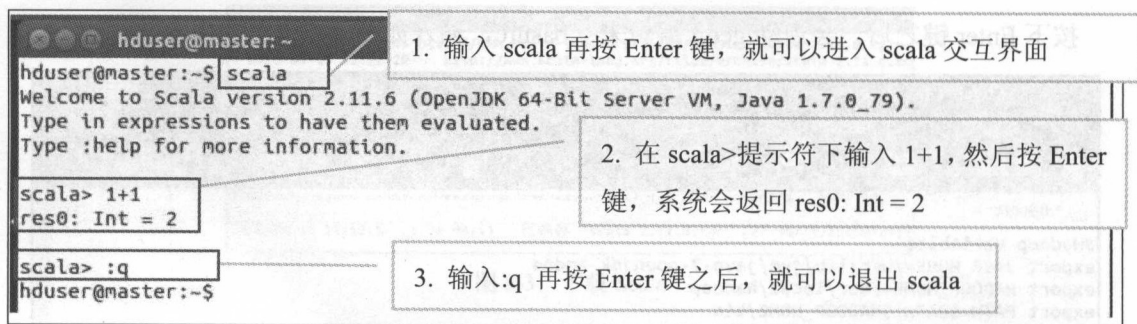


图 8-8 启动和退出 scala

8.3 安装 Spark

接下来介绍如何安装 Spark。

步骤 01 Spark 下载页面

在浏览器输入下列网址来下载 Spark：

```
https://spark.apache.org/downloads.html
```

注意！在此选择“choose a package type”，因为 Spark 会与 Hadoop 沟通，必须参照系统中目前的 Hadoop 版本进行选择，如图 8-9 所示。

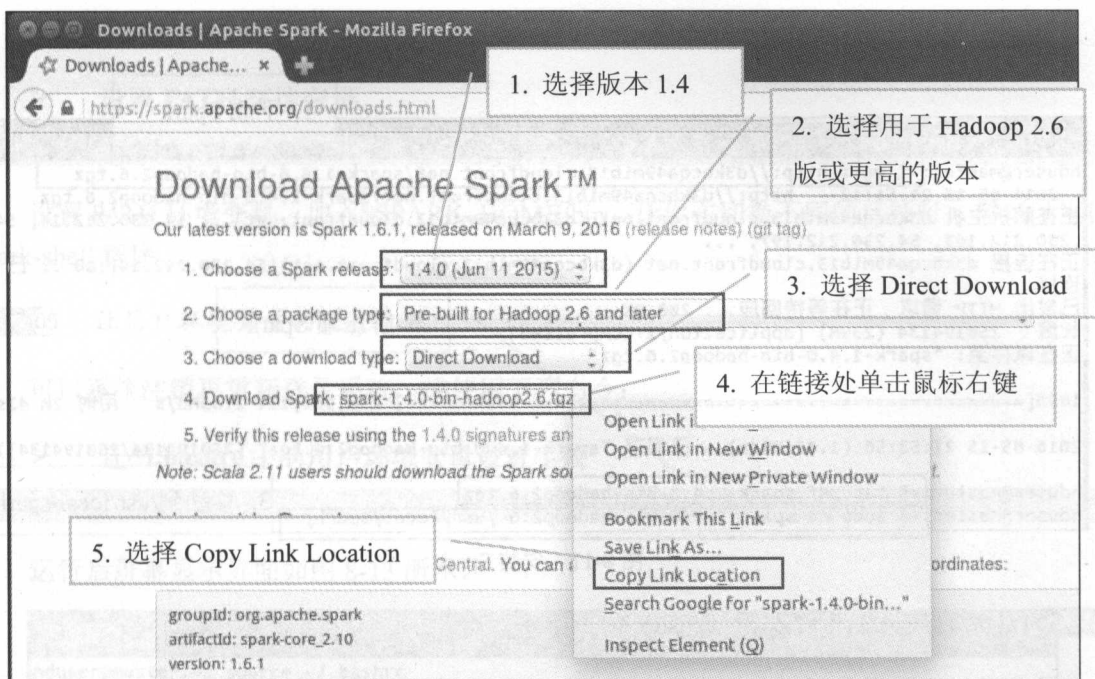


图 8-9 选择下载合适的 Spark 版本

步骤 02 粘贴链接，如图 8-10 所示。

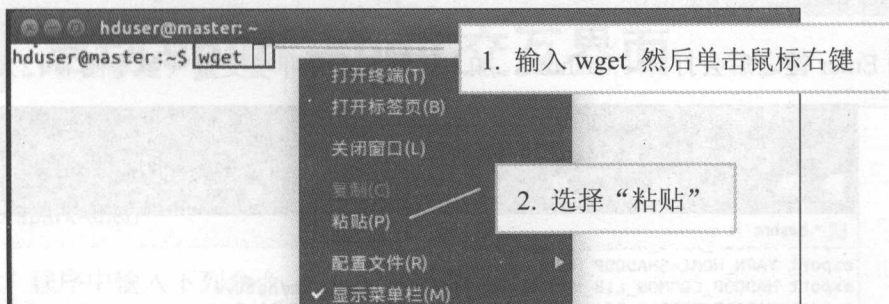


图 8-10 粘贴链接

步骤 03 安装 Spark

在“终端”程序中输入下列命令：

➤ 下载 Spark

```
wget http://d3kbcqa49mib13.cloudfront.net/spark-1.4.0-bin-hadoop2.6.tgz
```

➤ 解压缩 spark 到 spark-1.4.0-bin-hadoop2.6 目录

```
tar xzf spark-1.4.0-bin-hadoop2.6.tgz
```

➤ 把 spark-1.4.0-bin-hadoop2.6 目录移动到/usr/local/spark

```
sudo mv spark-1.4.0-bin-hadoop2.6 /usr/local/spark/
```

运行后屏幕显示界面如图 8-11 所示。

```

hduser@master: ~
hduser@master:~$ wget http://d3kbcqa49mib13.cloudfront.net/spark-1.4.0-bin-hadoop2.6.tgz
--2016-05-15 21:51:12-- http://d3kbcqa49mib13.cloudfront.net/spark-1.4.0-bin-hadoop2.6.tgz
正在解析主机 d3kbcqa49mib13.cloudfront.net (d3kbcqa49mib13.cloudfront.net)... 54.230.212.14, 54.230.212.161, 54.230.212.197, ...
正在连接 d3kbcqa49mib13.cloudfront.net (d3kbcqa49mib13.cloudfront.net)|54.230.212.14|:80... 已连接。
已发出 HTTP 请求，正在等待回应... 200 OK
长度：250194134 (239M) [application/x-compressed]
正在保存至：“spark-1.4.0-bin-hadoop2.6.tgz”

100%[=====] 250,194,134 2.05MB/s 用时 2m 43s
2016-05-15 21:53:56 (1.46 MB/s) - 已保存 “spark-1.4.0-bin-hadoop2.6.tgz” [250194134/250194134]

hduser@master ~$ tar xzf spark-1.4.0-bin-hadoop2.6.tgz
hduser@master ~$ sudo mv spark-1.4.0-bin-hadoop2.6 /usr/local/spark/
  
```

1. 下载 Spark

2. 解压缩 Spark

3. 移动到/usr/local/spark

图 8-11 安装 Spark

步骤 04 用户环境变量的设置 ~/.bashrc

在“终端”程序中输入下列命令：

➤ 编辑 ~/.bashrc

```
sudo gedit ~/.bashrc
```

按 Enter 键之后会打开 ~/.bashrc，加入下列 Spark 环境变量（参考图 8-12）。

```

*.bashrc (-) - gedit
文件(F) 编辑(E) 查看(V) 搜索(S) 工具(T) 文档(D) 帮助(H)

打开 保存 撤消

*.bashrc x
export YARN_HOME=$HADOOP_HOME
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib"
export JAVA_LIBRARY_PATH=$HADOOP_HOME/lib/native:$JAVA_LIBRARY_PATH

export PATH=${JAVA_HOME}/bin:${PATH}
export HADOOP_CLASSPATH=${JAVA_HOME}/lib/tools.jar

export SCALA_HOME=/usr/local/scala
export PATH=$PATH:$SCALA_HOME/bin

export SPARK_HOME=/usr/local/spark
export PATH=$PATH:$SPARK_HOME/bin
  
```

加入下列 Spark 环境变量

图 8-12 打开 ~/.bashrc 设置用户环境变量

上述命令说明如下：

➤ 设置 SPARK_HOME

```
export SPARK_HOME=/usr/local/spark
```

设置 SPARK_HOME，也就是 Spark 安装目录：/usr/local/spark。

➤ 设置 PATH 环境变量

```
export PATH=$PATH:$SPARK_HOME/bin
```

设置 PATH 环境变量，加入\$SPARK_HOME/bin，让你在切换不同的目录时，都可以执行 spark-shell 程序。

步骤 05 让用户环境变量的设置生效

可以系统注销再重新登录系统，或使用下列命令：

➤ 让 ~/.bashrc 中的用户环境变量设置生效

```
source ~/.bashrc
```

运行后屏幕显示界面如图 8-13 所示。

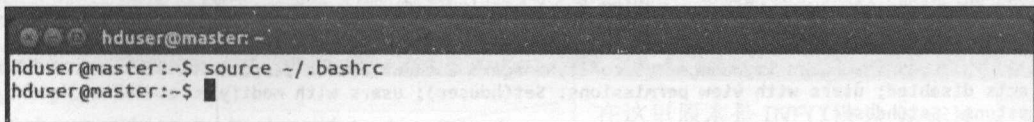


图 8-13 让 ~/.bashrc 中设置的用户环境变量生效

8.4 启动 spark-shell 交互界面

spark-shell 的优点是具有交互的好处，输入命令立刻就可以看到响应。

步骤 01 启动 spark-shell

在“终端”程序中输入下列命令，

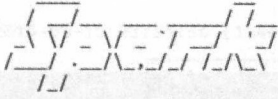
➤ 进入 spark 交互界面

```
spark-shell
```

运行后屏幕显示界面如图 8-14 所示，从中可以看到 Spark 与 Scala 的版本。

```

hduser@master: ~
hduser@master:~$ spark-shell
log4j:WARN No appenders could be found for logger (org.apache.hadoop.metrics2.
b.MutableMetricsFactory).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more i
fo.
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
15/06/29 23:32:38 INFO SecurityManager: Changing view acls to: hduser
15/06/29 23:32:38 INFO SecurityManager: Changing modify acls to: hduser
15/06/29 23:32:38 INFO SecurityManager: SecurityManager: authentication disable
; ui acls disabled; users with view permissions: Set(hduser); users with modify
permissions: Set(hduser)
15/06/29 23:32:38 INFO HttpServer: Starting HTTP Server
15/06/29 23:32:38 INFO Utils: Successfully started service 'HTTP class server'
n port 60211.
Welcome to

 version 1.4.0

Spark 版本

Using Scala version 2.10.4 (OpenJDK 64-Bit Server VM, Java 1.7.0_79)
Type in expressions to have them evaluated.
Type :help for more information.
15/06/29 23:32:44 INFO SparkContext: Running Spark version 1.4.0
15/06/29 23:32:45 INFO SecurityManager: Changing view acls to: hduser
15/06/29 23:32:45 INFO SecurityManager: Changing modify acls to: hduser
15/06/29 23:32:45 INFO SecurityManager: SecurityManager: authentication disable
; ui acls disabled; users with view permissions: Set(hduser); users with modify
permissions: Set(hduser)

Scala 版本

with 267.3 MB RAM, BlockManagerId(<driver>, localhost, 39268)
15/05/03 21:43:28 INFO BlockManagerMaster: Registered BlockManager
15/05/03 21:43:28 INFO SparkILoop: Created spark context.
Spark context available as sc.
15/05/03 21:43:28 INFO SparkILoop: Created sql context (with Hive support).
SQL context available as sqlContext.

最后会出现 scala>提
示符,这是可以输入命
令或输入 exit 退出

scala> exit
warning: there were 1 deprecation warning(s); re-run with -deprecation for details
hduser@master:/usr/local/spark$

```

图 8-14 启动 spark-shell

8.5 设置 spark-shell 显示信息

Spark 系统安装后,在 spark-shell 交互界面中默认会显示很多信息。有时太多信息会影响我们的阅读,所以建议修改设置,让它只显示警告信息。

步骤 01 复制 log4j 模板文件

在“终端”程序中输入下列命令:

➤ 切换到 spark 配置文件目录

```
cd /usr/local/spark/conf
```

➤ 复制 log4j 模板文件到 log4j.properties

```
cp log4j.properties.template log4j.properties
```


运行后屏幕显示界面如图 8-15 所示。

```
hduser@master: /usr/local/spark/conf
hduser@master:~$ cd /usr/local/spark/conf
hduser@master:/usr/local/spark/conf$ cp log4j.properties.template log4j.properties
```

图 8-15 复制 log4j 模板文件

步骤 02 设置 log4j

在“终端”程序中输入下列命令：

➤ 编辑 log4j.properties

```
sudo gedit log4j.properties
```

按 Enter 键之后会打开 log4j.properties，屏幕显示界面如图 8-16 所示。

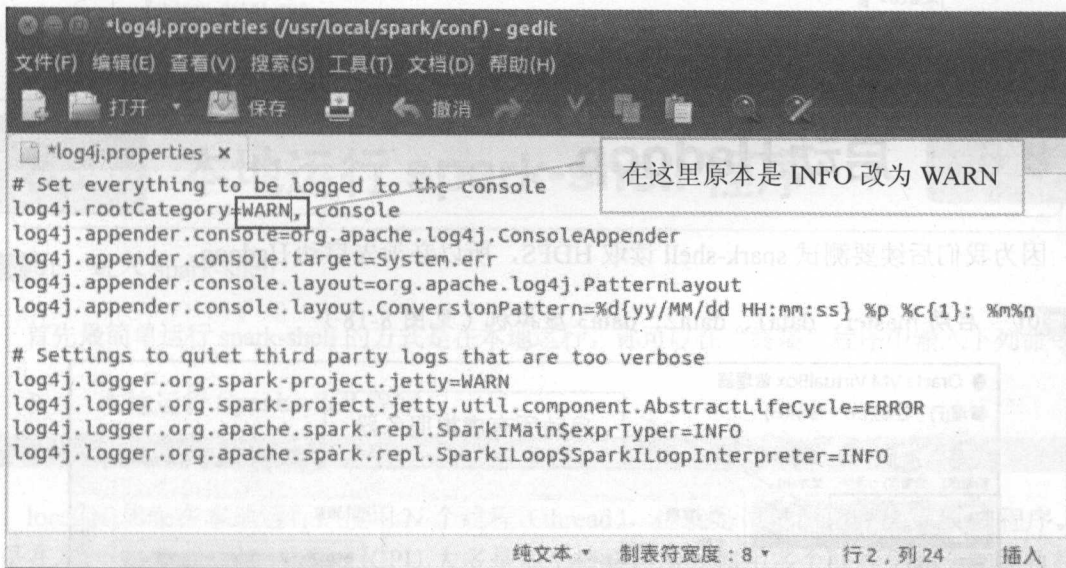


图 8-16 设置 log4j

参照图 8-16，将 log4j.rootCategory 原本的 INFO（信息）改为 WARN（警告），然后单击“保存”按钮关闭窗口。

步骤 03 再次进入 spark-shell

再次在“终端”程序中输入下列命令：

➤ 启动 spark-shell

```
spark-shell
```

你会发现信息少了很多，只列出了重要部分的信息，如图 8-17 所示。

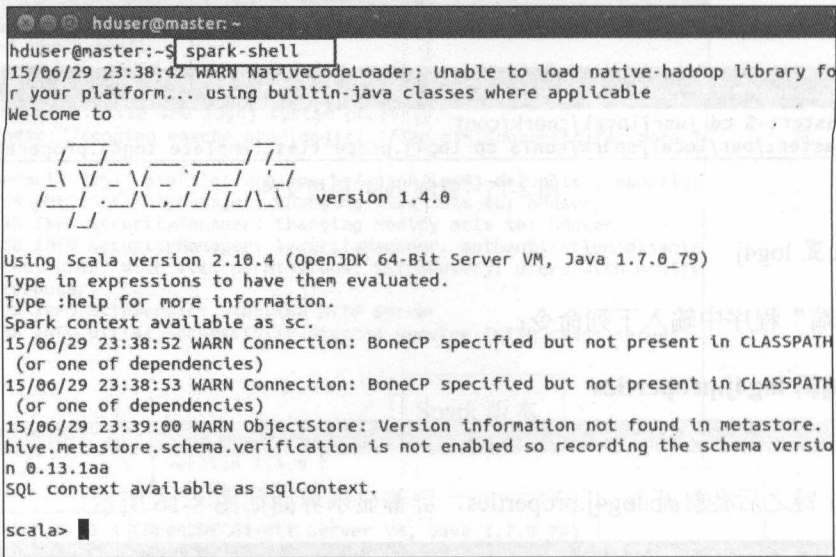


图 8-17 再次进入 spark-shell，信息少了很多了

8.6 启动 Hadoop

因为我们后续要测试 spark-shell 读取 HDFS，所以必须先启动 Hadoop。

步骤 01 启动 master, data1, data2, data3 虚拟机（见图 8-18）

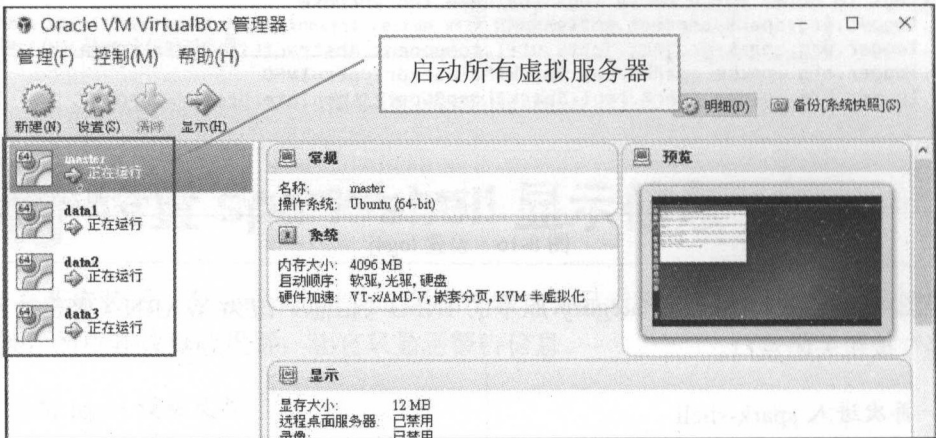


图 8-18 启动 master, data1, data2, data3 虚拟机

步骤 02 启动 Hadoop

在 master 虚拟机，启动“终端”程序。
输入下列命令启动 Hadoop multi nodes cluster:

```
start-all.sh
```

土地年租金 5—10 元/亩

```
spark-shell--master local[4]
```

运行后屏幕显示界面如图 8-20 所示。

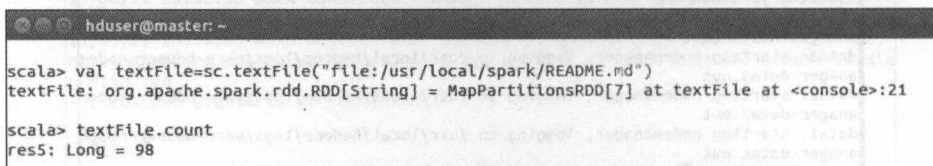
步骤 02 读取本地文件

接下来我们将执行一个简单的 Spark 程序，在 spark-shell 提示符 `scala>` 后输入下列命令：

➤ 读取本地的文件

```
val textFile=sc.textFile("file:/usr/local/spark/README.md")
textFile.count
```

以上命令在路径前加上“file:”，告诉系统就是要读取本地文件，并且列出数据记录数，运行后屏幕显示界面如图 8-21 所示。



```
hduser@master: ~
scala> val textFile=sc.textFile("file:/usr/local/spark/README.md")
textFile: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[7] at textFile at <console>:21
scala> textFile.count
res5: Long = 98
```

图 8-21 读取本地文件

步骤 03 读取 HDFS 文件

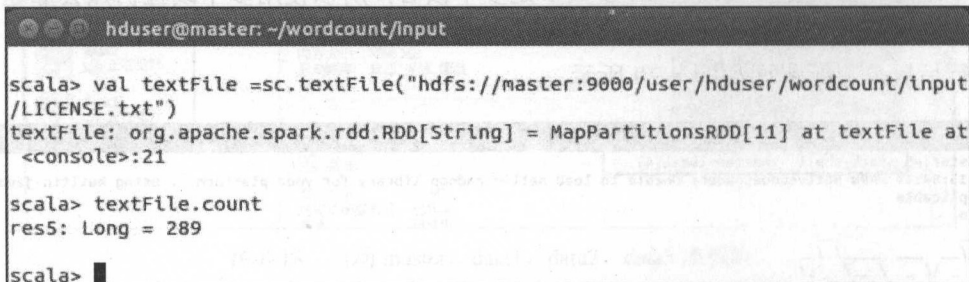
你也可以使用 `sc.textFile` 读取 HDFS 文件，如下列范例。读取第 7.4 小节上传至 HDFS 的文本文件 `pg5000.txt`。

在 spark-shell 提示符 `scala>` 后输入下列命令：

➤ 读取 HDFS 文件

```
val textFile =sc.textFile("hdfs://master:9000/user/hduser/wordcount/input/LICENSE.txt")
textFile.count
```

以上命令，在路径前加上“`hdfs://master:9000`”，告诉系统要读取 HDFS 文件。执行后屏幕显示界面如图 8-22 所示。



```
hduser@master: ~/wordcount/input
scala> val textFile =sc.textFile("hdfs://master:9000/user/hduser/wordcount/input/LICENSE.txt")
textFile: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[11] at textFile at <console>:21
scala> textFile.count
res5: Long = 289
scala>
```

图 8-22 读取 HDFS 文件

以上 HDFS 存取路径“`hdfs://master:9000`”，是我们在安装 Hadoop 时设置的，请参考第 5.2 节步骤 5 “编辑 `core-site.xml`”。

8.8 在 Hadoop YARN 运行 spark-shell

Spark 可以在 Hadoop YARN 上运行, 让 YARN 帮助它进行多台机器资源的管理。接下来, 介绍如何在 Hadoop YARN 运行 spark-shell。

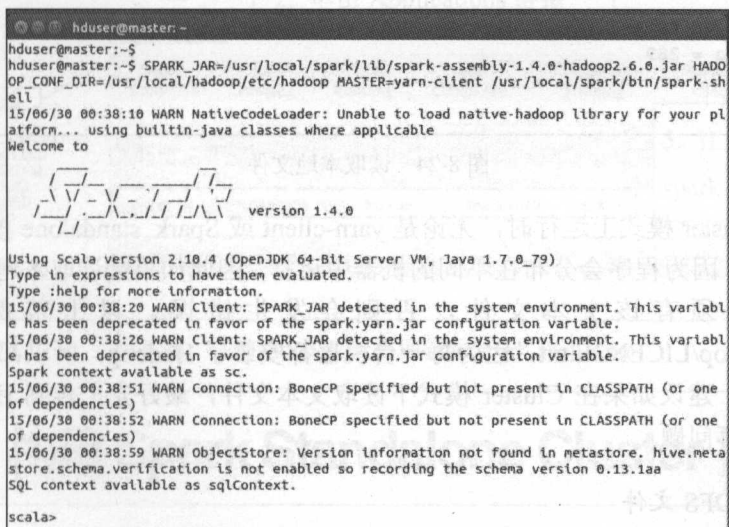
步骤 01 在 Hadoop YARN 运行 spark-shell

在 master 虚拟机启动“终端”程序, 输入下列命令:

➤ 在 Hadoop YARN 运行 spark-shell

```
SPARK_JAR=/usr/local/spark/lib/spark-assembly-1.4.0-hadoop2.6.0.jar
HADOOP_CONF_DIR=/usr/local/hadoop/etc/Hadoop MASTER=yarn-client /usr/local/
spark/bin/spark-shell
```

运行后, 屏幕显示界面如图 8-23 所示, 最后会出现 scala>提示符。



```
hduser@master: ~
hduser@master:~$ SPARK_JAR=/usr/local/spark/lib/spark-assembly-1.4.0-hadoop2.6.0.jar HADOOP_CONF_DIR=/usr/local/hadoop/etc/hadoop MASTER=yarn-client /usr/local/spark/bin/spark-shell
15/06/30 00:38:10 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Welcome to

  ____      _
 / ___|    / \
| |  | |  / _ \
| |  | | / ___ \
| |  | || |___) |
| |  | || |___) |
| |  | || |___) |
|_|  |_| \____/

version 1.4.0

Using Scala version 2.10.4 (OpenJDK 64-Bit Server VM, Java 1.7.0_79)
Type in expressions to have them evaluated.
Type :help for more information.
15/06/30 00:38:20 WARN Client: SPARK_JAR detected in the system environment. This variable has been deprecated in favor of the spark.yarn.jar configuration variable.
15/06/30 00:38:26 WARN Client: SPARK_JAR detected in the system environment. This variable has been deprecated in favor of the spark.yarn.jar configuration variable.
Spark context available as sc.
15/06/30 00:38:51 WARN Connection: BoneCP specified but not present in CLASSPATH (or one of dependencies)
15/06/30 00:38:52 WARN Connection: BoneCP specified but not present in CLASSPATH (or one of dependencies)
15/06/30 00:38:59 WARN ObjectStore: Version Information not found in metastore. hive.meta.store.schema.version is not enabled so recording the schema version 0.13.1aa
SQL context available as sqlContext.

scala>
```

图 8-23 在 Hadoop YARN 运行 spark-shell

以上命令详细说明如下:

➤ 设置 Spark jar 文件路径

```
SPARK_JAR=/usr/local/spark/lib/spark-assembly-1.4.0-hadoop2.6.0.jar
```

➤ 设置 Hadoop 配置文件目录

```
HADOOP_CONF_DIR=/usr/local/hadoop/etc/hadoop
```

➤ 设置运行模式是 yarn-client

```
MASTER=yarn-client
```

➤ 要运行的程序 spark-shell 的完整路径

```
/usr/local/spark/bin/spark-shell
```

步骤 02 读取本地文件

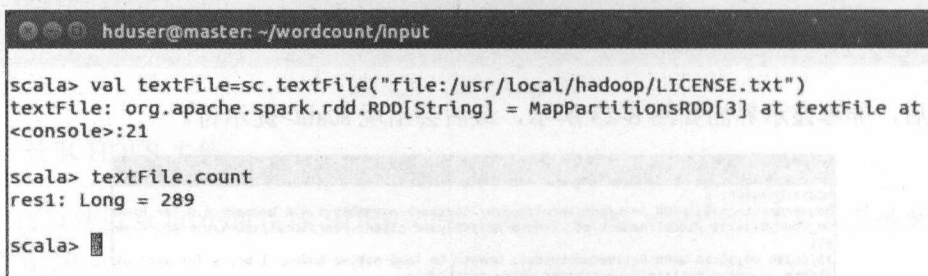
在 scala>提示符后输入下列命令：

➤ 读取本地文件

```
val textFile=sc.textFile("file:/usr/local/hadoop/LICENSE.txt")
textFile.count
```

在路径前加上“file:”，告诉系统要读取本地文件。

运行结果如图 8-24 所示。



```
hduser@master: ~/wordcount/input

scala> val textFile=sc.textFile("file:/usr/local/hadoop/LICENSE.txt")
textFile: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[3] at textFile at
<console>:21

scala> textFile.count
res1: Long = 289

scala> █
```

图 8-24 读取本地文件

当用户在 Cluster 模式上运行时，无论是 yarn-client 或 Spark standalone 模式，运行 Spark 读取文本文件时，因为程序会分布在不同的机器中运行，因此如果是读取本地文件，请确认所有的机器都必须有该文本文件，否则会发生错误。以上范例，因为读取 file:/usr/local/hadoop/LICENSE.txt，因为每一台机器都安装了 Hadoop，所以都会有该文件，所以不会发生错误。建议如果在 Cluster 模式下读取文本文件，最好先上传到 HDFS，再读取文本文件，就不会有问题。

步骤 03 读取 HDFS 文件

在 scala>提示符后输入下列命令：

➤ 读取 HDFS 文件

路径前加上“hdfs://master:9000”，告诉系统要读取 HDFS 文件。

```
val textFile = sc.textFile("hdfs://master:9000/user/hduser/wordcount/
input/LICENSE.txt")
```

➤ 显示记录数

```
textFile.count
```

运行结果如图 8-25 所示。

```
hduser@master: ~
scala> val textFile = sc.textFile("hdfs://master:9000/user/hduser/wordcount/input/pg5000.txt")
textFile: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[3] at textFile at <console>:21

scala> textFile.count()
res1: Long = 32116
```

图 8-25 读取 HDFS 文件

步骤 04 在 Hadoop Web 界面可以查看 spark-shell App

现在我们已经 Hadoop YARN 运行了 spark-shell，所以就可以在 Hadoop Web 界面看到这个应用程序（Application）。

参照下列步骤打开 Hadoop Web 界面（参见图 8-26）。

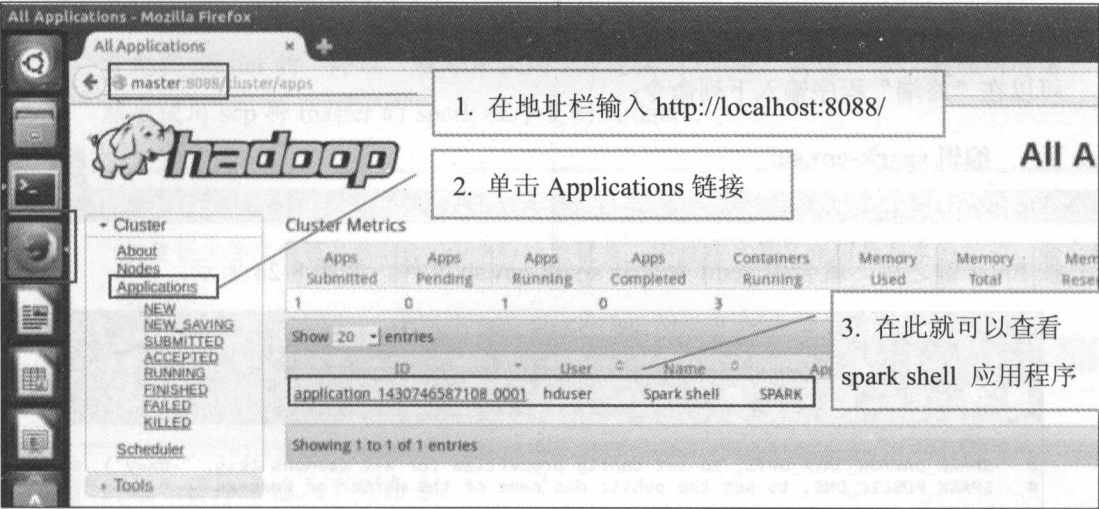


图 8-26 在 Hadoop Web 界面查看 spark-shell App

8.9 构建 Spark Standalone Cluster 执行环境

接下来，我们将构建 Spark Standalone Cluster 运行环境，然后测试 spark-shell 在 Spark Standalone Cluster 环境中的运行。具体步骤如下：

构建步骤	说明
1. 在 master 虚拟机设置 spark-env.sh	设置 Spark Standalone Worker 使用的 CPU、内存等
2. 复制 spark 程序到 data1、data2、data3	将 master 的 spark 程序，复制到 data1、data2、data3
3. 在 master 虚拟机编辑 slaves 文件	设置 Spark Standalone Cluster 有哪些服务器

步骤 01 复制模板文件（template）来创建 spark-env.sh

spark-env.sh 是 Spark 的环境配置文件。Spark 系统中提供了模板文件，更便于用户设置时

作为参照。在 master 虚拟机启动“终端”程序，输入下列命令：

➤ 复制模板文件来创建 spark-env.sh

```
cp /usr/local/spark/conf/spark-env.sh.template /usr/local/spark/conf/spark-env.sh
```

运行后，屏幕显示界面如图 8-27 所示。

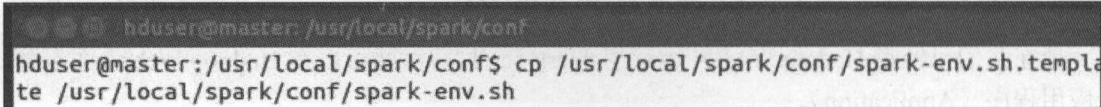


图 8-27 复制模板文件（template）来创建 spark-env.sh

步骤 02 设置 spark-env.sh

可以在“终端”程序输入下列命令：

➤ 编辑 spark-env.sh:

```
sudo gedit /usr/local/spark/conf/spark-env.sh
```

按 Enter 键之后，就会用 gedit 来打开 spark-env.sh 文件，如图 8-28 所示。

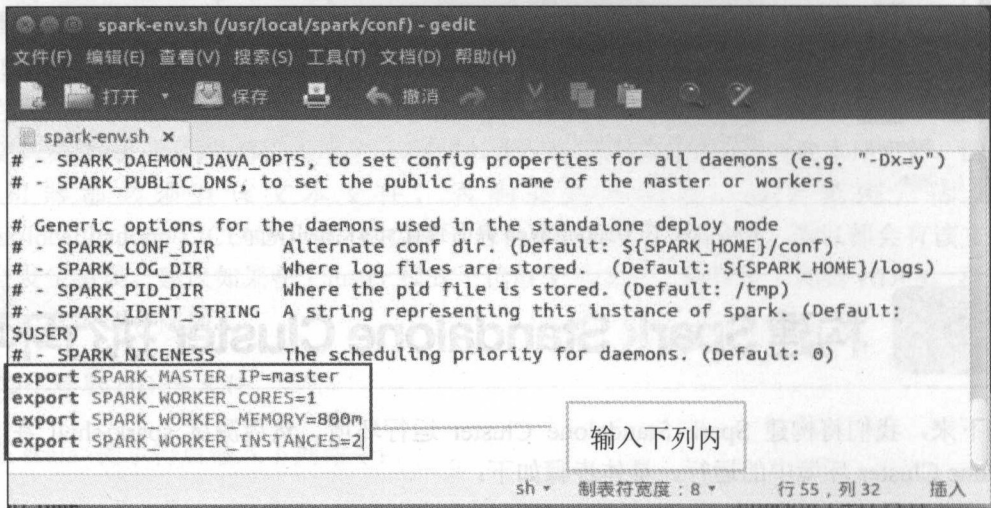


图 8-28 编辑 spark-env.sh 进行相应的设置

spark-env.sh 文件的设置内容，说明如下：

➤ 设置 master 的 IP 或服务器名称

```
export SPARK_MASTER_IP=master
```

➤ 设置每个 Worker 使用的 CPU 核心

```
export SPARK_WORKER_CORES=1
```


➤ 设置每个 Worker 使用内存

```
export SPARK_WORKER_MEMORY=800m
```

内存的设置，必须考虑所使用计算机的内存容量，如果计算机的容量不足，则不要设置太多。

➤ 设置每个 Worker 实例

```
export SPARK_WORKER_INSTANCES=2
```

如果 Host（即主机）的内存不足，则设置为 1 就好。

步骤 03 将 master 的 spark 程序复制到 data1

接下来，我们将 master 的 spark 程序复制到 data1，具体步骤如下：

- 首先在 master 的“终端”程序中使用 ssh 连接到 data1，创建目录、更改所有者。
- 然后使用 scp 将 master 的 spark 程序复制到 data1。

提示

scp（是 secure copy 的缩写），可在 Linux 加密传输、进行远程文件复制，以及从本地主机复制文件到远程主机。scp 命令的功能很多，不过在这里只介绍最基本的功能，命令格式如下：

```
scp -r [本地文件] [远程用户名称]@[远程主机名] : [远程目录]
```

-r 递归复制整个目录。

在“终端”程序中输入下列命令：

➤ SSH 连接至 data1

```
ssh data1
```

➤ 连接成功后，创建 spark 目录

```
sudo mkdir /usr/local/spark
```

➤ 更改所有者为 hduser

```
sudo chown hduser:hduser /usr/local/spark
```

➤ 注销 data1

```
exit
```

➤ 使用 scp 将 master 的 spark 程序复制到 data1

```
sudo scp -r /usr/local/spark hduser@data1:/usr/local
```

复制前系统会询问 data1 密码，请输入密码。

运行后，屏幕显示界面如图 8-29 所示。

```

hduser@master: ~
hduser@master:~$ ssh data1
Welcome to Ubuntu 14.10 (GNU/Linux 3.16.0-23-generic x86_64)

* Documentation:  https://help.ubuntu.com/

145 packages can be updated.
145 updates are security updates.

New release '15.04' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Tue May  5 00:34:15 2015 from master
hduser@data1:~$ sudo mkdir /usr/local/spark
[sudo] password for hduser:
hduser@data1:~$ sudo chown hduser:hduser /usr/local/spark
hduser@data1:~$ exit
logout
Connection to data1 closed.
hduser@master:~$ sudo scp -r /usr/local/spark hduser@data1:/usr/local
[sudo] password for hduser:
hduser@data1's password:
sample_fpgrowth.txt 100% 68 0.1KB/s 00:00
lpsa.data            100% 10KB 10.2KB/s 00:00
lr_data.txt          100% 192KB 192.5KB/s 00:00
  
```

图 8-29 将 master 的 spark 程序复制到 data1

步骤 04 将 master 的 spark 程序复制到 data2

使用下列命令，通过 ssh 与 scp 将 master 的 spark 程序复制到 data2。

在“终端”程序中输入下列命令：

➤ SSH 连接至 data2

```
ssh data2
```

➤ 连接成功后，创建 spark 目录

```
sudo mkdir /usr/local/spark
```

➤ 更改所有者为 hduser

```
sudo chown hduser:hduser /usr/local/spark
```

➤ 注销 data2

```
exit
```

➤ 使用 scp 将 master 的 spark 程序复制到 data2

```
sudo scp -r /usr/local/spark hduser@data2:/usr/local
```

复制前系统会询问 data2 密码，请输入密码。

运行后，屏幕显示界面如图 8-30 所示。

```

hduser@master: ~
hduser@master:~$ ssh data2
Welcome to Ubuntu 14.10 (GNU/Linux 3.16.0-23-generic x86_64)

 * Documentation:  https://help.ubuntu.com/
146 packages can be updated.
146 updates are security updates.

New release '15.04' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Tue May  5 00:44:49 2015 from master
hduser@data2:~$ sudo mkdir /usr/local/spark
[sudo] password for hduser:
hduser@data2:~$ sudo chown hduser:hduser /usr/local/spark
hduser@data2:~$ exit
logout
Connection to data2 closed.
hduser@master:~$ sudo scp -r /usr/local/spark hduser@data2:/usr/local
hduser@data2's password:
sample_fpgrowth.txt          100% 68      0.1KB/s   00:00
lpsa.data                   100% 10KB    10.2KB/s   00:00
lr_data.txt                  100% 192KB   192.5KB/s   00:00
sample_isotonic_regression_data.txt 100% 1598    1.6KB/s   00:00

```

1. SSH 连接至 data2

2. 连接成功后创建 spark 目录

3. 更改所有者为 hduser

4. 注销 data2

5. scp 从 master 复制到 data2

图 8-30 将 master 的 spark 程序复制到 data2

步骤 05 将 master 的 spark 程序复制到 data3

使用下列命令，通过 ssh 与 scp 将 master 的 spark 程序复制到 data3。
在“终端”程序中输入下列命令：

➤ SSH 连接至 data3

```
ssh data3
```

➤ 连接成功后，创建 spark 目录

```
sudo mkdir /usr/local/spark
```

➤ 更改所有者为 hduser

```
sudo chown hduser:hduser /usr/local/spark
```

➤ 注销 data3

```
exit
```

➤ 使用 scp 将 master 的 spark 程序复制到 data3

```
sudo scp -r /usr/local/spark hduser@data3:/usr/local
```

复制前系统会询问 data3 密码，请输入密码。
运行后，屏幕显示界面如图 8-31 所示。

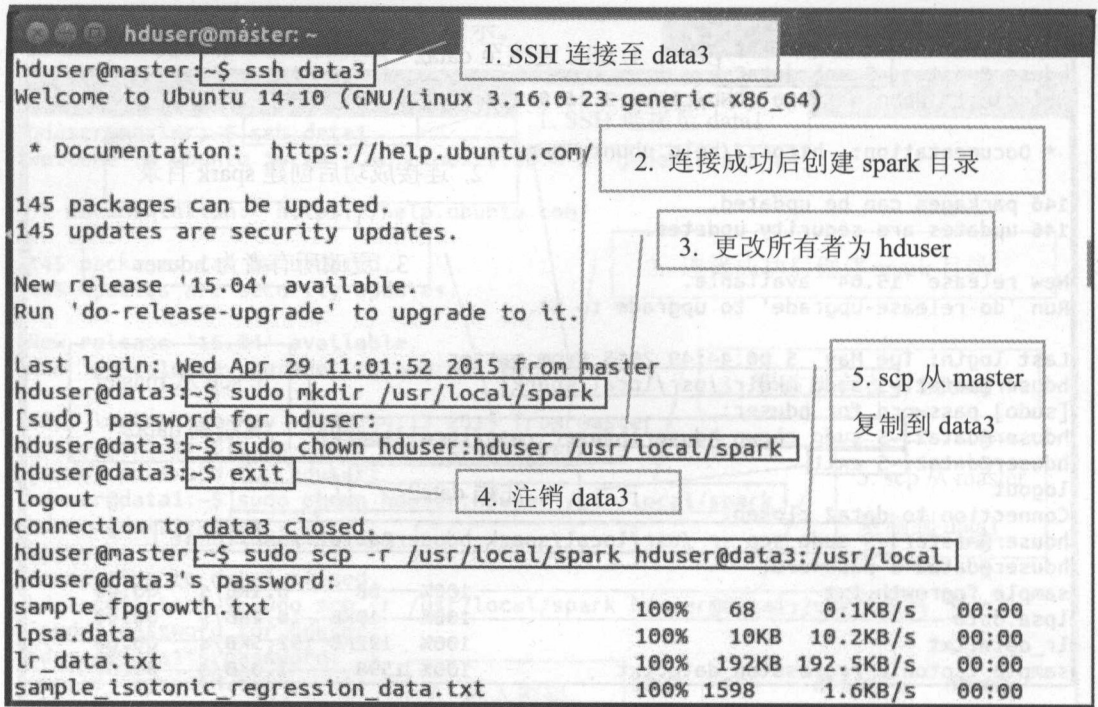


图 8-31 将 master 的 spark 程序复制到 data3

步骤 06 编辑 slaves 文件

可以在“终端”程序中输入下列命令：

➤ 编辑 slaves 文件，设置 Spark Standalone Cluster 有哪些服务器

```
sudo gedit /usr/local/spark/conf/slaves
```

按 Enter 键之后，就会用 gedit 打开 slaves 文件，如图 8-32 所示。

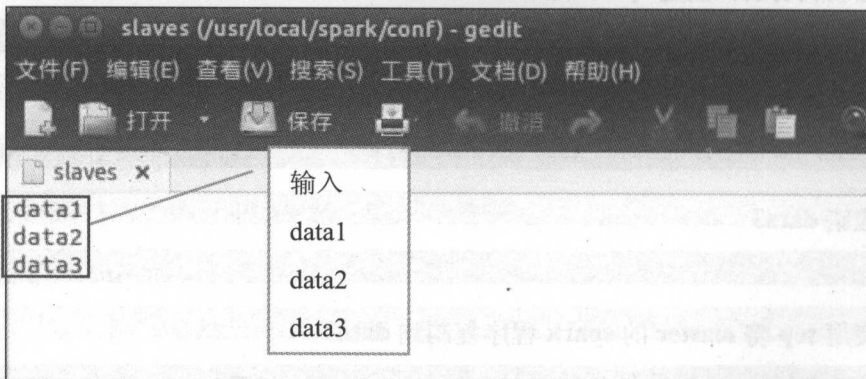


图 8-32 编辑 slaves 文件

8.10 在 Spark Standalone 运行 spark-shell

步骤 01 启动 Spark Standalone Cluster

可以在“终端”程序中输入下列命令：

➤ 启动 Spark Standalone Cluster

```
/usr/local/spark/sbin/start-all.sh
```

运行后屏幕显示界面如图 8-33 所示，可以看到一共启动了 6 个 worker。

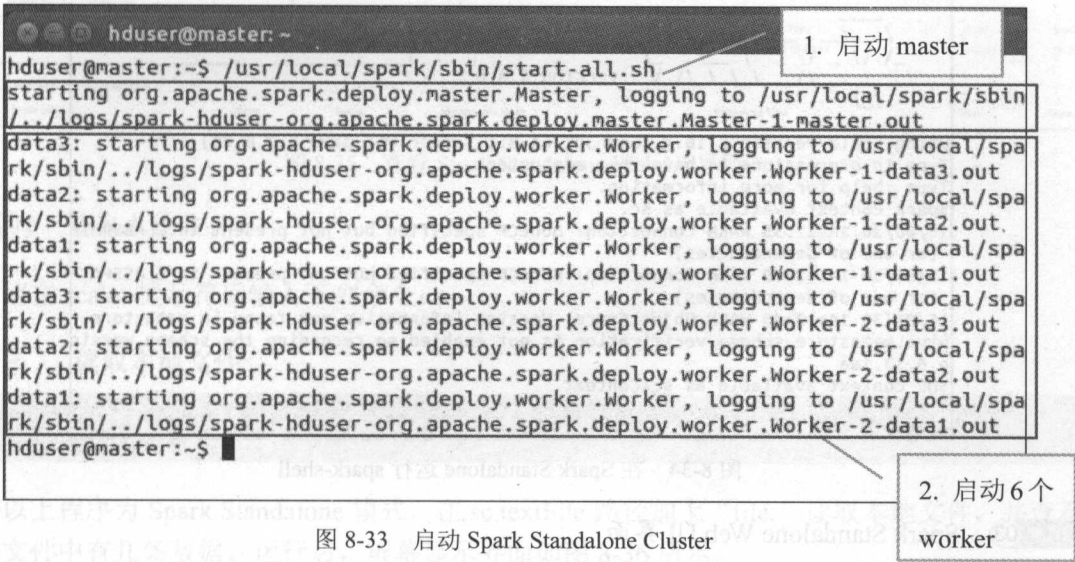


图 8-33 启动 Spark Standalone Cluster

因为我们之前在配置中进行了下列设置：

- 在 slaves 文件中，设置有 3 个服务器 data1、data2、data3。
- 在 spark-env.sh 文件中，设置 SPARK_WORKER_INSTANCES=2。

所以 3*2=6，在此显示出 worker 共有 6 个。

➤ 分别启动 master 与 slaves

之前我们使用 start-all.sh 会同时启动 master 与 slaves。也可以分别启动 master 与 slaves，如下列命令：

命令	说明
/usr/local/spark/sbin/start-master.sh	启动 master 服务器
/usr/local/spark/sbin/start-slaves.sh	启动 slaves 服务器

步骤 02 在 Spark Standalone 运行 Spark-shell

可以在“终端”程序中输入下列命令:

在 Spark Standalone 运行 spark-shell 程序

```
spark-shell --master spark://master:7077
```

运行后屏幕显示界面如图 8-34 所示。

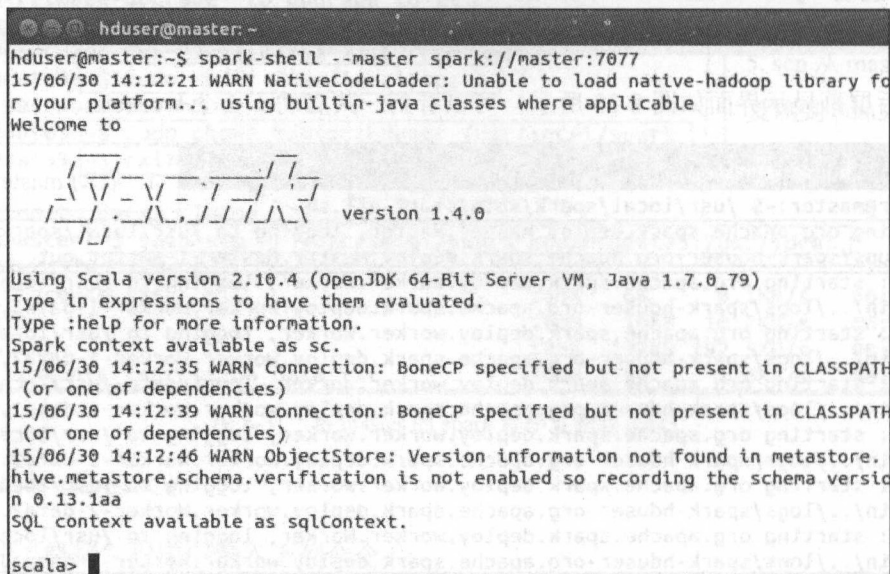


图 8-34 在 Spark Standalone 运行 spark-shell

步骤 03 Spark Standalone Web UI 界面

可以在浏览器输入下列网址:

查看 Spark Standalone Web UI 界面

http://master:8080/

运行后屏幕显示界面如图 8-35 所示,从中可以看到启动后共有 6 个 worker 与当前运行的程序 spark-shell:

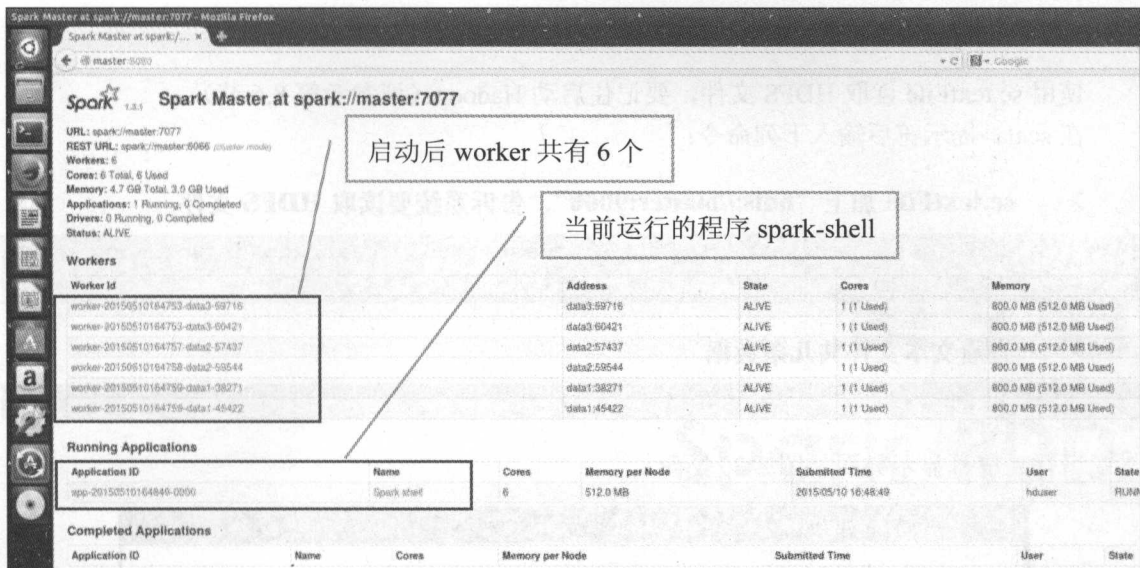


图 8-35 查看 Spark Standalone Web UI 界面

步骤 04 读取本地文件

请在 `scala>` 提示符后输入下列命令:

➤ 读取本地文件

```
val textFile=sc.textFile("file:/usr/local/hadoop/LICENSE.txt")
textFile.count
```

以上程序为 Spark Standalone 模式, 在 `sc.textFile` 路径加上 “file:” 读取本地文件, 并查看文本文件中几条数据。运行后, 屏幕显示界面如图 8-36 所示。

```
hduser@master: ~/wordcount/input
scala> val textFile=sc.textFile("file:/usr/local/hadoop/LICENSE.txt")
textFile: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[1] at textFile at
<console>:21

scala> textFile.count
res0: Long = 289
```

图 8-36 读取本地文件

当在 Cluster 模式下运行时, 无论是 `yarn-client` 或 Spark Standalone 模式, 运行 Spark 读取文本文件时, 因为程序会分布在不同的机器中执行, 所以如果是读取本地文件, 请确认所有的机器都必须有该文本文件, 否则会发生错误。以上范例, 我们要读取 `file:/usr/local/hadoop/LICENSE.txt`, 因为每一台机器都安装了 Hadoop, 都会有该文件, 所以不会发生错误。建议: 如果在 Cluster 模式读取文本文件, 最好先把文件上传到 HDFS, 再读取文本文件, 就不会有问题。

步骤 05 读取 HDFS 文件

使用 `sc.textFile` 读取 HDFS 文件，要记住启动 Hadoop（请参考第 8.6 节）。
在 `scala>` 提示符后输入下列命令：

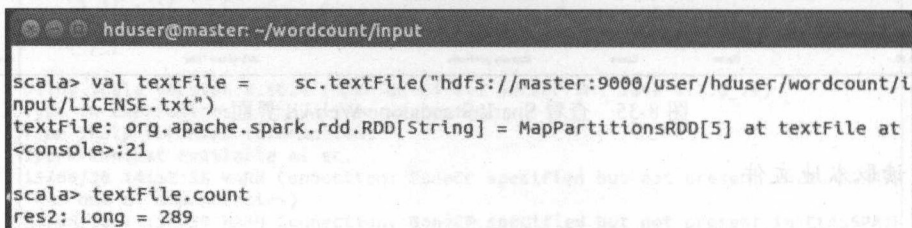
➤ **`sc.textFile` 加上 “`hdfs://master:9000`”，告诉系统要读取 HDFS 文件**

```
val textFile = sc.textFile("hdfs://master:9000/user/hduser/wordcount/
input/LICENSE.txt")
```

➤ **查看文本文件共几条数据**

```
textFile.count
```

运行后屏幕显示界面如图 8-37 所示。



```
hduser@master: ~/wordcount/input
scala> val textFile = sc.textFile("hdfs://master:9000/user/hduser/wordcount/
input/LICENSE.txt")
textFile: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[5] at textFile at
<console>:21
scala> textFile.count
res2: Long = 289
```

图 8-37 读取 HDFS 文件

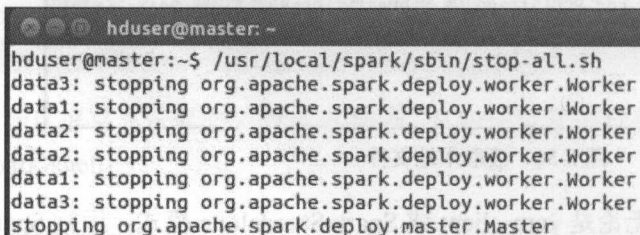
步骤 06 停止 Spark stand alone cluster

最后在“终端”程序中输入下列命令：

➤ **停止 Spark stand alone cluster**

```
/usr/local/spark/sbin/stop-all.sh
```

运行后屏幕显示界面如图 8-38 所示。



```
hduser@master: ~
hduser@master:~$ /usr/local/spark/sbin/stop-all.sh
data3: stopping org.apache.spark.deploy.worker.Worker
data1: stopping org.apache.spark.deploy.worker.Worker
data2: stopping org.apache.spark.deploy.worker.Worker
data2: stopping org.apache.spark.deploy.worker.Worker
data1: stopping org.apache.spark.deploy.worker.Worker
data3: stopping org.apache.spark.deploy.worker.Worker
stopping org.apache.spark.deploy.master.Master
```

图 8-38 停止 Spark standalone cluster

第 9 章

Spark RDD

- 9.1 RDD 的特性
- 9.2 基本 RDD “转换” 运算
- 9.3 多个 RDD “转换” 运算
- 9.4 基本 “动作” 运算
- 9.5 RDD Key-Value 基本 “转换” 运算
- 9.6 多个 RDD Key-Value “转换” 运算
- 9.7 Key-Value “动作” 运算
- 9.8 Broadcast 广播变量
- 9.9 accumulator 累加器
- 9.10 RDD Persistence 持久化
- 9.11 使用 Spark 创建 WordCount
- 9.12 Spark WordCount 详细解说

Spark 的核心是 RDD (Resilient Distributed Dataset)，即弹性分布式数据集，是由 AMPLab 实验室所提出的概念，属于一种分布式的内存系统的数据集应用。Spark 主要优势就是来自 RDD 本身的特性。RDD 能与其他系统兼容，可以导入外部存储系统的数据集，例如：HDFS、HBase 或其他 Hadoop 数据源。

9.1 RDD 的特性

➤ RDD 的三种基本运算

在 RDD 之上，可以施加三种类型的运算：

RDD 运算类型	说明
“转换”运算 Transformation	RDD 执行“转换”运算的结果，会产生另外一个 RDD 但是由于 RDD 的 lazy 特性，“转换”运算并不会立刻实际执行，它会等到执行到“动作”运算，才会实际执行
“动作”运算 Action	RDD 执行“动作”运算后，不会产生另外一个 RDD，它会产生数值、数组或写入文件系统 RDD 执行“动作”运算时会立刻实际执行，并且连同之前的“转换”运算一起执行
“持久化” Persistence	对于那些会重复使用的 RDD，可以将 RDD “持久化”在内存中作为后续使用，以提高执行性能

RDD 通过“转换”运算可以得出新的 RDD，但 Spark 会延迟这个“转换”动作的发生时间点。它并不会马上执行，而是等到了执行了 Action 之后，才会基于所有的 RDD 关系来执行转换。示意图如图 9-1 所示。

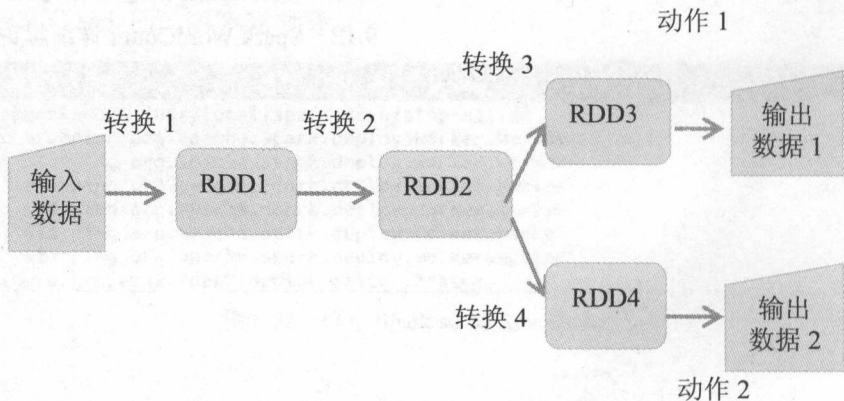


图 9-1 RDD 运算示意图

如图 9-1 所示中的说明如下：

- 输入数据，执行“转换 1”运算产生 RDD1，此时不会实际执行，只记录操作命令。
- RDD1 执行“转换 2”运算产生 RDD2，此时不会实际执行，只记录操作命令。
- RDD2 执行“转换 3”运算产生 RDD3，此时不会实际执行，只记录操作命令。
- RDD2 执行“转换 4”运算产生 RDD4，此时不会实际执行，只记录操作命令。
- RDD3 执行“动作 1”运算，此时会实际执行：“转换 1”+“转换 2”+“转换 3”+“动作 1”，产生输出数据 1。
- RDD4 执行“动作 2”运算，此时会实际执行：“转换 1”+“转换 2”+“转换 4”+“动作 2”，产生输出数据 2。
- (如果你之前已经先执行了“动作 1”运算，所以“转换 1”+“转换 2”已经实际执行完成，在此只会实际执行“转换 4”+“动作 2”，以节省运行时间)。

➤ Lineage 机制具备容错的特性

RDD 本身具有 Lineage 机制，它会记录每个 RDD 与其父代 RDD 之间的关联，它会记录通过什么操作，才由父代 RDD 得到该 RDD 的信息。

RDD 本身的 immutable 不可变的特性，再加上 Lineage 机制，使得 Spark 具备容错的特性。如果某节点的机器出现了故障，那么存储于这个节点上的 RDD 损毁了，于是会重新执行一连串的“转换”命令，产生新的输出数据，如此就可以避免因为特定节点的故障，造成整个系统无法运行的问题。

➤ RDD 命令整理

以下 RDD 命令已整理在本书的博客文章中。当练习安装时，可以复制博客文章中的命令，然后粘贴到“终端”程序中。这样可节省打字的时间，也不用担心打错字(如果无法在 VirtualBox 虚拟机的 Ubuntu“终端”程序中执行复制/粘贴操作时，参考第 3.9 节的说明，设置好 VirtualBox 的共享剪贴板)。本书博客的网址为：

<http://blog.sina.com.cn/hadoopsparkbook>

9.2 基本 RDD “转换”运算

步骤 01 进入 spark-shell

在“终端”程序中输入下列命令：

➤ 进入 spark-shell 交互界面

```
spark-shell
```

执行后就会出现 scala>提示符，这是就可以开始输入命令了。后续介绍的 Spark 命令，都是在 scala>提示符后输入，如图 9-2 所示。


```

hduser@master:~$ spark-shell
15/06/30 21:43:34 WARN NativeCodeLoader: Unable to load native-hadoop library for
your platform... using builtin-java classes where applicable
Welcome to

      _/  __|  _/  _/
     /_  /  _/  _/  _/  version 1.4.0

Using Scala version 2.10.4 (OpenJDK 64-Bit Server VM, Java 1.7.0_79)
Type in expressions to have them evaluated.
Type :help for more information.
Spark context available as sc.
15/06/30 21:43:47 WARN Connection: BoneCP specified but not present in CLASSPATH
(or one of dependencies)
15/06/30 21:43:48 WARN Connection: BoneCP specified but not present in CLASSPATH
(or one of dependencies)
15/06/30 21:43:54 WARN ObjectStore: Version information not found in metastore.
hive.metastore.schema.verification is not enabled so recording the schema version
0.13.1aa
SQL context available as sqlContext.

scala>

```

图 9-2 进入 spark-shell 交互界面

步骤 02 创建 intRDD

创建 RDD 最简单的方式，就是使用 SparkContext 的 `parallelize` 方法，如下列命令：

➤ 创建 intRDD

```
val intRDD = sc.parallelize(List(3,1, 2, 5, 5))
```

先使用 `val` 定义 `intRDD`，然后使用 `parallelize` 方法输入一个 `List` 的参数，以创建 `intRDD`。不过这这也是一个“转换”运算，所以不会马上实际执行。

➤ intRDD 转换为 Array

```
intRDD.collect()
```

`intRDD` 执行 `collect()` 后，会转换为 `Array`。这是一个“动作”运算，所以会立刻执行，执行后返回 `Array`，屏幕显示界面如图 9-3 所示。

```

hduser@master:~$
scala> val intRDD = sc.parallelize(List(3,1, 2, 5, 5))
intRDD: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[0] at parallelize
at <console>:21

scala> intRDD.collect()
res0: Array[Int] = Array(3, 1, 2, 5, 5)

```

图 9-3 创建 intRDD

提示

在 Scala 语言中，声明常数与变量方式如下：

- `val` 声明常数——声明后就不能修改
- `var` 声明变量——声明后仍然可以修改

Scala 语言鼓励程序设计人员尽量使用 `val` 来声明常数，尽量不要使用 `var` 声明变量。

因为在 Scala 语言设计哲学中，认为在程序流程中修改变量的值，会造成程序逻辑复杂、不易维护、不容易实现并行计算。

步骤 03 创建 stringRDD

parallelize 方法除了可以创建 Int 的 RDD，也可以创建 String 的 RDD，如下列命令：

➤ 创建 stringRDD

```
val stringRDD = sc.parallelize(List("Apple", "Orange", "Banana", "Grape", "Apple"))
```

➤ 将 stringRDD 转换为 Array

```
stringRDD.collect()
```

运行后，屏幕显示界面如图 9-4 所示。

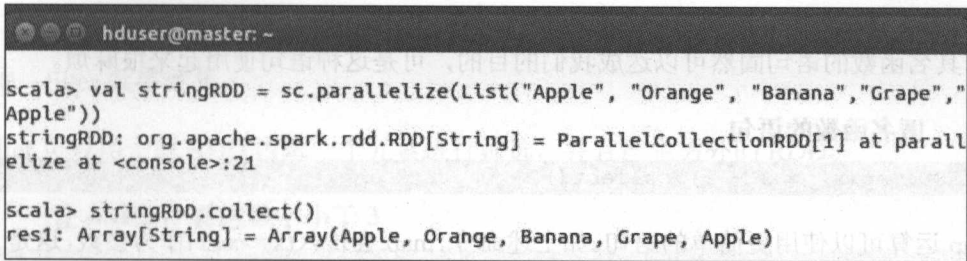


图 9-4 创建 stringRDD

步骤 04 map 运算

map 运算可以通过传入的函数，将每一个元素经过函数运算产生另外一个 RDD。如图 9-5 所示，RDD 通过传入的函数 addOne，将每一个元素加 1 而产生另外一个 RDD。

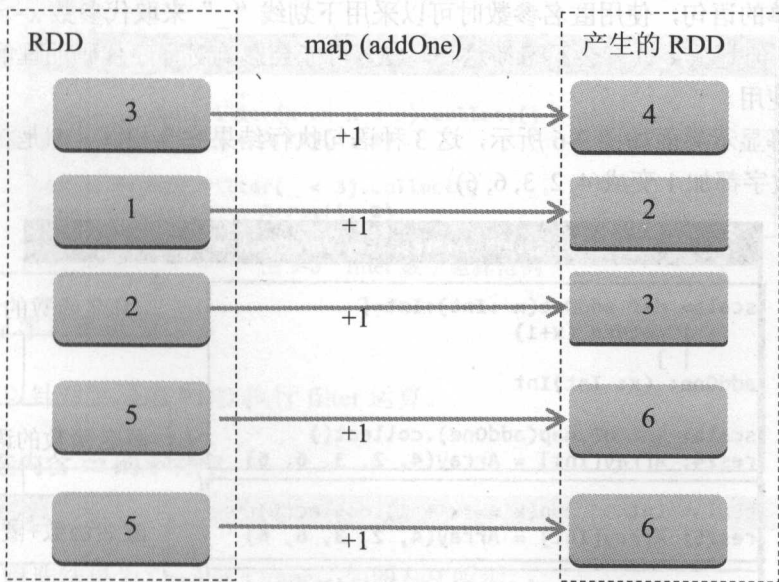


图 9-5 map 运算示意图

在 spark 的 map 运算中，可以使用下列 3 种语句：

➤ 具名函数的语句

```
def addOne(x :Int):Int={
  return (x+1)
}
intRDD.map(addOne).collect()
```

1. 先定义 addOne 函数并传入参数 x，此函数会将 x 加 1 再返回。
2. 然后将函数名称 addOne 作为参数传入 map 命令，map 命令会将每一个元素加 1，从而产生另外一个 RDD。
3. 不过因为 map 是一个“转换”运算，所以不会马上执行。为了方便示范，所以我们加上 collect() 这个“动作”运算使之立即执行。
4. 具名函数的语句固然可以达成我们的目的，可是这种语句使用起来很麻烦。

➤ 匿名函数的语句

```
intRDD.map(x => x + 1).collect()
```

map 运算可以使用更简单的语句。如上述命令，map 会传入 (x=>x+1) 作为参数，这是 lambda 语句的 anonymous functions 匿名函数。其中 x 是传入参数，x+1 是要执行的命令，告诉 map 运算每一个元素都要加 1。匿名函数的语句简洁多了，而且让程序代码更易读。

➤ 匿名函数 + 匿名参数的语句

```
intRDD.map(_ + 1).collect()
```

还有更简单的语句，使用匿名参数时可以采用下划线“_”来取代参数 x=>x+1。这种语句固然更简洁，但是很多读者会不习惯这种写法，可能会影响对程序代码的理解，因此可以视个人喜好选择使用。

运行后屏幕显示界面如图 9-6 所示，这 3 种语句执行结果完全相同，都是将原本(3,1,2,5,5)中的每一个数字都加 1 变成(4,2,3,6,6)。

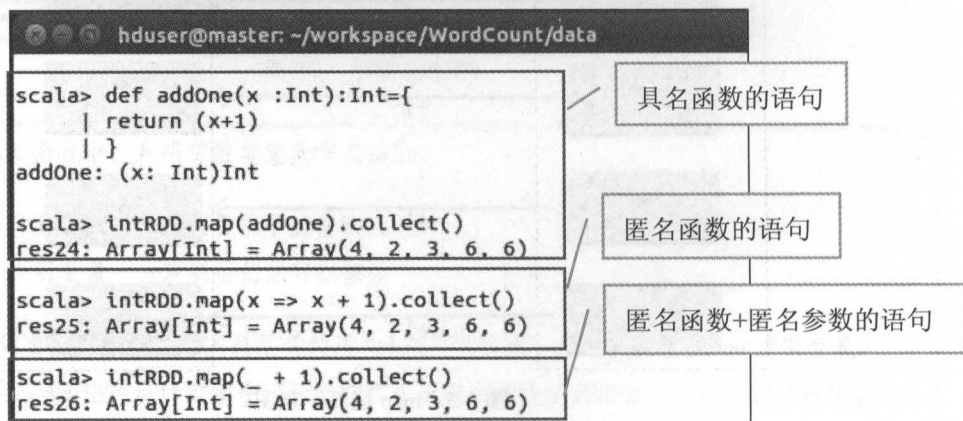


图 9-6 map 运算的 3 种实现语句，输出结果完全相同

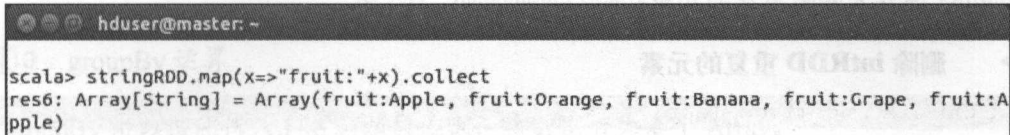
步骤 05 map 字符串运算

我们也可以针对字符串 RDD 执行 map 运算。

➤ 将 **stringRDD** 所有字符串元素前面加上 **fruit:**，来产生另一个 RDD

```
stringRDD.map(x=>"fruit:"+x).collect
```

运行结果如图 9-7 所示。



```
hduser@master: ~
scala> stringRDD.map(x=>"fruit:"+x).collect
res6: Array[String] = Array(fruit:Apple, fruit:Orange, fruit:Banana, fruit:Grape, fruit:Apple)
```

图 9-7 map 字符串运算

步骤 06 filter 数字运算

filter 可以用于对 RDD 内每一个元素进行筛选，并且产生另外的 RDD。

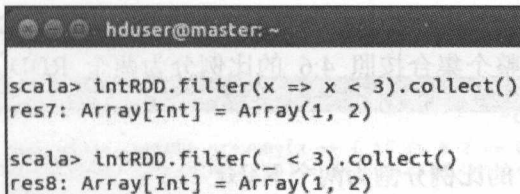
➤ 让 **intRDD** 筛选数字小于 3

```
intRDD.filter(x => x < 3).collect()
```

➤ 使用匿名参数让筛选数字小于 3，可使用下划线 “_” 来取代 **x=>x < 3**

```
intRDD.filter(_ < 3).collect()
```

以上两条语句的运行结果完全相同，将原本(3,1, 2, 5, 5) 筛选小于 3 的数字，筛选结果为 (1, 2)。如图 9-8 所示。



```
hduser@master: ~
scala> intRDD.filter(x => x < 3).collect()
res7: Array[Int] = Array(1, 2)

scala> intRDD.filter(_ < 3).collect()
res8: Array[Int] = Array(1, 2)
```

图 9-8 filter 数字运算范例

步骤 07 filter 字符串运算

我们也可以针对字符串 RDD 执行 filter 运算。

➤ 筛选内含 **ra** 的字符串

```
stringRDD.filter(x =>x.contains("ra") ).collect()
```

运行后，筛选结果为 Orange, Grape，如图 9-9 所示。

```
hduser@master: ~
scala> stringRDD.filter(x =>x.contains("ra")).collect()
res9: Array[String] = Array(Orange, Grape)
```

图 9-9 filter 字符串运算范例

步骤 08 distinct 运算

distinct 运算会删除重复的元素。

➤ 删除 intRDD 重复的元素

```
intRDD.distinct().collect()
```

➤ 删除 stringRDD 重复的元素

```
stringRDD.distinct().collect()
```

运行后会删除重复的元素，如图 9-10 所示。

```
hduser@master: ~
scala> intRDD.distinct().collect()
res15: Array[Int] = Array(1, 3, 5, 2)

scala> stringRDD.distinct().collect()
res16: Array[String] = Array(Grape, Orange, Apple, Banana)
```

图 9-10 distinct 运算范例

步骤 09 randomSplit 运算

randomSplit 可以将整个集合元素，以随机数的方式按照比例分为多个 RDD。

如下命令使用 randomSplit 将整个集合按照 4:6 的比例分为两个 RDD。此运算会返回 Array[org.apache.spark.rdd.RDD[Int]]。

➤ 以随机数的方式按照 4:6 的比例分割为两个 RDD

```
val sRDD = intRDD.randomSplit(Array(0.4,0.6))
```

运行后返回 sRDD 数组，可以使用下列命令：

➤ 读取分割后的 2 个 RDD

命令	说明
sRDD(0).collect()	使用 sRDD(0)读取第 1 个 RDD
sRDD(1).collect()	sRDD(1)读取第 2 个 RDD

运行结果的屏幕显示界面如图 9-11 所示。


```

hduser@master: ~
scala> val sRDD = intRDD.randomSplit(Array(0.4,0.6))
sRDD: Array[org.apache.spark.rdd.RDD[Int]] = Array(PartitionwiseSampledRDD[80] at randomSplit at <console>:23, PartitionwiseSampledRDD[81] at randomSplit at <console>:23)

scala> sRDD(0).collect()
res82: Array[Int] = Array(3, 5)

scala> sRDD(1).collect()
res83: Array[Int] = Array(1, 2, 5)

```

图 9-11 randomSplit 运算范例

步骤 10 groupBy 运算

groupBy 可以按照传入的匿名函数规则，将数据分为多个 Array。

➤ 使用 groupBy 运算将整个集合分成奇数与偶数

使用 groupBy 运算时，传入的匿名函数 $x \Rightarrow \{ \text{if } (x \% 2 == 0) \text{"even"} \text{ else "odd"} \}$ ，将整个集合按照奇数与偶数分为两个 Array，此运算会返回 `Array[(String, Iterable[Int])]`。

```

val gRDD=intRDD.groupBy(
x =>{ if (x % 2 == 0) "even" else "odd"}
).collect

```

➤ 读取第 1 个是偶数的 Array

```
gRDD(0)
```

➤ 读取第 2 个是奇数的 Array

```
gRDD(1)
```

运行结果的屏幕显示界面如图 9-12 所示。

```

hduser@master: ~
scala> val gRDD= intRDD.groupBy(x => { if (x % 2 == 0) "even" else "odd" }).collect
gRDD: Array[(String, Iterable[Int])] = Array((even,CompactBuffer(2)), (odd,CompactBuffer(3, 1, 5, 5)))

scala> gRDD(0)
res91: (String, Iterable[Int]) = (even,CompactBuffer(2))

scala> gRDD(1)
res92: (String, Iterable[Int]) = (odd,CompactBuffer(3, 1, 5, 5))

```

图 9-12 groupBy 运算范例

9.3 多个 RDD “转换” 运算

RDD 也支持执行多个 RDD 的运算。

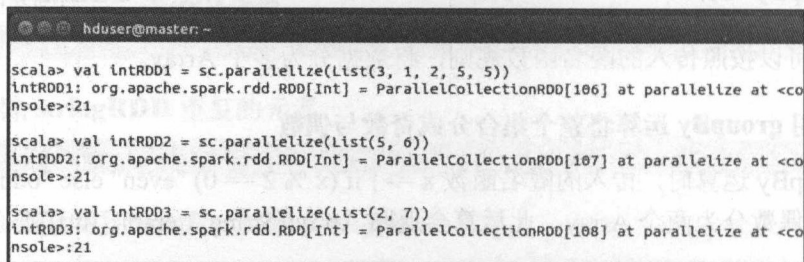
步骤 01 创建 3 个范例 RDD

为了示范多个 RDD 的运算，我们使用下列命令：

➤ 创建 3 个范例 RDD

```
val intrRDD1 = sc.parallelize(List(3, 1, 2, 5, 5))
val intrRDD2 = sc.parallelize(List(5, 6))
val intrRDD3 = sc.parallelize(List(2, 7))
```

运行后，屏幕显示界面如图 9-13 所示。



```
hduser@master: ~
scala> val intrRDD1 = sc.parallelize(List(3, 1, 2, 5, 5))
intrRDD1: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[106] at parallelize at <console>:21

scala> val intrRDD2 = sc.parallelize(List(5, 6))
intrRDD2: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[107] at parallelize at <console>:21

scala> val intrRDD3 = sc.parallelize(List(2, 7))
intrRDD3: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[108] at parallelize at <console>:21
```

图 9-13 创建 3 个范例 RDD

步骤 02 union 并集运算

可以使用下列命令，将 intrRDD1、intrRDD2、intrRDD3 进行并集运算。

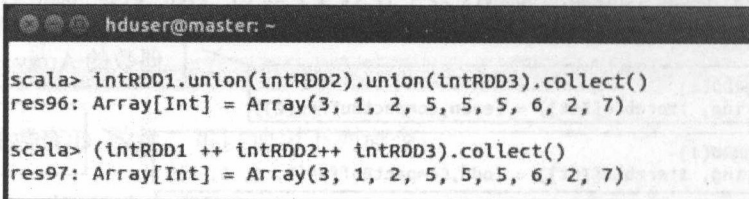
➤ 使用 union 函数进行并集运算

```
intrRDD1.union(intrRDD2).union(intrRDD3).collect()
```

➤ 使用 ++ 符号进行并集运算

```
(intrRDD1 ++ intrRDD2 ++ intrRDD3).collect()
```

运行后屏幕显示界面如图 9-14 所示，这两条语句的结果完全相同。



```
hduser@master: ~
scala> intrRDD1.union(intrRDD2).union(intrRDD3).collect()
res96: Array[Int] = Array(3, 1, 2, 5, 5, 5, 6, 2, 7)

scala> (intrRDD1 ++ intrRDD2 ++ intrRDD3).collect()
res97: Array[Int] = Array(3, 1, 2, 5, 5, 5, 6, 2, 7)
```

图 9-14 union 并集运算范例

步骤 03 intersection 交集运算

可以使用下列命令进行运算：

➤ 将 intrRDD1、intrRDD2 进行交集运算

```
intrRDD1.intersection(intrRDD2).collect()
```

intRDD1 是 List(3, 1, 2, 5, 5)，与 intRDD2 的 List(5, 6)之间的重复元素只有 5，所以返回 Array(5)。运行后的结果如图 9-15 所示。

```
scala> intRDD1.intersection(intRDD2).collect()
res98: Array[Int] = Array(5)
```

图 9-15 intersection 交集运算范例

步骤 04 subtract 差集运算

可以使用下列命令进行运算：

➤ 进行差集运算

```
intRDD1.subtract(intRDD2).collect()
```

intRDD1 是 List(3, 1, 2, 5, 5)。扣除 intRDD2 List(5, 6)重复的部分 5，所以结果是(1,2,3)。运行后的结果如图 9-16 所示。

```
scala> intRDD1.subtract(intRDD2).collect()
res99: Array[Int] = Array(1, 2, 3)
```

图 9-16 subtract 差集运算范例

步骤 05 cartesian 笛卡尔乘积运算

可以使用下列命令进行运算：

➤ 进行 cartesian 笛卡尔乘积运算

```
intRDD1.cartesian(intRDD2).collect()
```

intRDD1 有 5 个元素、intRDD2 有两个元素，所以笛卡尔乘积运算后，会产生 $5 \times 2 = 10$ 组的数据。运行后的结果如图 9-17 所示。

```
scala> intRDD1.cartesian(intRDD2).collect()
res101: Array[(Int, Int)] = Array((3,5), (3,6), (1,5), (1,6), (2,5), (2,6), (5,5), (5,6), (5,5), (5,6))
```

图 9-17 cartesian 笛卡尔乘积运算范例

9.4 基本“动作”运算

步骤 01 读取元素

可以使用下列命令读取 RDD 内的元素。这是 Actions 运算，所以会马上执行：

➤ 读取第 1 条数据

```
intRDD.first
```

➤ 读取前几条数据，例如读取前 2 条

```
intRDD.take(2)
```

➤ 按照从小到大排序读取前 N 条数据，例如读取前 3 条

```
intRDD.takeOrdered(3)
```

➤ 按照从大到小排序读取前 N 条数据，例如读取前 3 条

```
intRDD.takeOrdered(3)(Ordering[Int].reverse)
```

运行后的结果如图 9-18 所示。

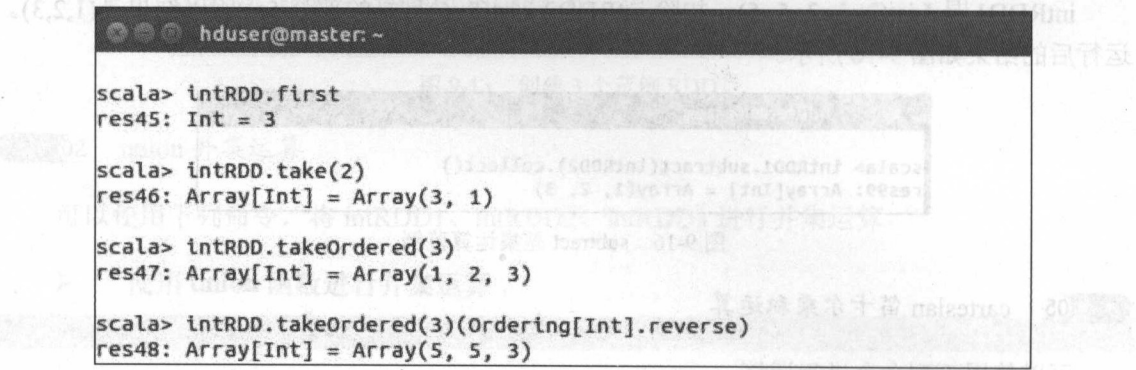


图 9-18 读取元素范例

步骤 02 统计功能

还可以将 RDD 内的元素进行统计运算。

命令	说明
intRDD.stats	统计
intRDD.min	最小
intRDD.max	最大
intRDD.stdev	标准偏差
intRDD.count	计数
intRDD.sum	求和
intRDD.mean	平均

运行后的结果如图 9-19 所示。


```
hduser@master: ~
scala> intRDD.stats
res61: org.apache.spark.util.StatCounter = (count: 5, mean: 3.200000, stdev: 1.600000, max: 5.000000, min: 1.000000)

scala> intRDD.min
res62: Int = 1

scala> intRDD.max
res63: Int = 5

scala> intRDD.stdev
res64: Double = 1.6

scala> intRDD.count
res65: Long = 5

scala> intRDD.sum
res66: Double = 16.0

scala> intRDD.mean
res67: Double = 3.2
```

图 9-19 统计功能范例

9.5 RDD Key-Value 基本“转换”运算

Spark RDD 支持键值（Key-Value）运算，Key-Value 运算也是 Map/Reduce 的基础。本节将介绍 RDD 键值的基本“转换”运算。

步骤 01 创建范例 Key-Value RDD

为了示范 RDD Key-Value 基本 Transformation 运算，先使用下列命令：

➤ 创建范例 Key-Value RDD

```
val kvRDD1 = sc.parallelize(List((3, 4), (3, 6), (5, 6), (1, 2)))
```

运行后，屏幕显示界面如图 9-20 所示。

```
hduser@master: ~
scala> val kvRDD1 = sc.parallelize(List((3, 4), (3, 6), (5, 6), (1, 2)))
kvRDD1: org.apache.spark.rdd.RDD[(Int, Int)] = ParallelCollectionRDD[2] at parallelize at <console>:21
```

图 9-20 创建范例 Key-Value RDD

运行后，会创建下列 Key-ValueRDD：

KEY	VALUE
3	4
3	6
5	6
1	2

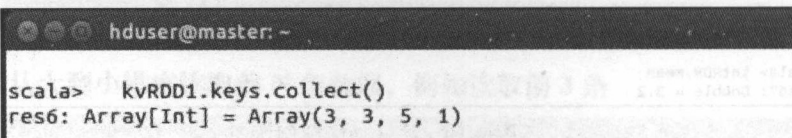
步骤 02 列出 keys 值

可以使用下列命令：

➤ 列出 kvRDD1 所有的 Key 值

```
kvRDD1.keys.collect()
```

以我们的测试数据为例：List((3, 4), (3, 6), (5, 6), (1, 2))的第一个字段是 Key，所有的 Key 是(3,3,5,1)。运行后屏幕显示界面如图 9-21 所示。



```
hduser@master: ~  
scala> kvRDD1.keys.collect()  
res6: Array[Int] = Array(3, 3, 5, 1)
```

图 9-21 列出 keys 值范例

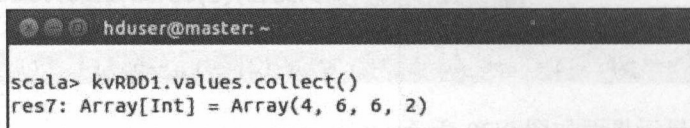
步骤 03 列出 values 值

可以使用下列命令：

➤ 列出所有 kvRDD1 的 values

```
kvRDD1.values.collect()
```

运行后屏幕显示界面如图 9-22 所示。以我们的测试数据为例：List((3, 4), (3, 6), (5, 6), (1, 2))的第二个字段是 values，所有的 values 是(4,6,6,2)。



```
hduser@master: ~  
scala> kvRDD1.values.collect()  
res7: Array[Int] = Array(4, 6, 6, 2)
```

图 9-22 列出 values 值范例

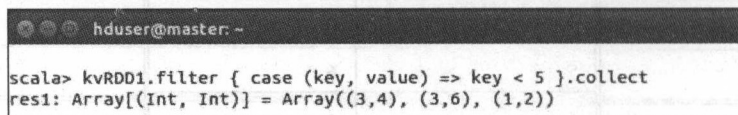
步骤 04 使用 filter 筛选 key 运算

可以使用 filter，针对 key 筛选 RDD 内的元素。如下列命令：

➤ 使用 filter 筛选出 key<5

```
kvRDD1.filter { case (key, value) => key < 5 }.collect
```

以我们的测试数据为例：(3, 4), (3, 6), (5, 6), (1, 2)的 key（第一个字段）小于 5，共有 3 条数据(3,4),(3,6),(1,2)。运行后的结果如图 9-23 所示。



```
hduser@master: ~  
scala> kvRDD1.filter { case (key, value) => key < 5 }.collect  
res1: Array[(Int, Int)] = Array((3,4), (3,6), (1,2))
```

图 9-23 使用 filter 筛选 key 运算的范例

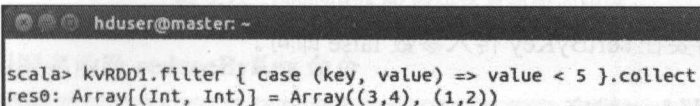
步骤 05 使用 filter 筛选 value 运算

可以使用 filter 针对 value 筛选出 RDD 内的元素。如下列命令：

➤ 使用 filter 筛选出 value<5

```
kvRDD1.filter{ case (key, value) => value < 5 }.collect
```

以我们的测试数据为例：(3, 4), (3, 6), (5, 6), (1, 2) 的 value（第二个字段）小于 5，共有两条数据(3,4),(1,2)。执行后的屏幕显示界面如图 9-24 所示。



```
hduser@master: ~
scala> kvRDD1.filter { case (key, value) => value < 5 }.collect
res0: Array[(Int, Int)] = Array((3,4), (1,2))
```

图 9-24 使用 filter 筛选 value 运算的范例

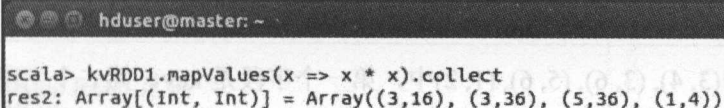
步骤 06 mapValues 运算

mapValues 运算可以针对 RDD 内每一组(Key,Value)进行运算，并且产生另外一个 RDD。如下列命令：

➤ 将每一组(Key,Value)的 value 值进行平方运算

```
kvRDD1.mapValues(x => x * x).collect
```

以我们的测试数据为例：(3, 4), (3, 6), (5, 6), (1, 2) 以 value（第二个字段）进行平方运算，也就是(3, 4*4), (3, 6*6), (5, 6*6), (1, 2*2)；结果是(3,16),(3,36),(5,36),(1,4)。运行后屏幕显示界面如图 9-25 所示。



```
hduser@master: ~
scala> kvRDD1.mapValues(x => x * x).collect
res2: Array[(Int, Int)] = Array((3,16), (3,36), (5,36), (1,4))
```

图 9-25 mapValues 运算的范例

步骤 07 sortByKey 从小到大按照 key 排序

可以使用 sortByKey() 按照 key 排序。传入参数默认值是 true，也就是从小到大排序。

➤ 从小到大按照 key 排序

```
kvRDD1.sortByKey(true).collect()
```

➤ 因为参数默认是 true，所以可以不传入参数

```
kvRDD1.sortByKey().collect()
```

屏幕显示界面如图 9-26 所示，以上两条语句的执行结果完全相同。

```
hduser@master: ~  
scala> kvRDD1.sortByKey(true).collect()  
res10: Array[(Int, Int)] = Array((1,2), (3,4), (3,6), (5,6))  
  
scala> kvRDD1.sortByKey().collect()  
res8: Array[(Int, Int)] = Array((1,2), (3,4), (3,6), (5,6))
```

图 9-26 sortByKey 从小到大按照 key 排序的范例

步骤 08 sortByKey 从大到小按照 key 排序

在本步骤中，只需要在 sortByKey 传入参数 false 即可。

➤ 从大到小按照 key 排序

```
kvRDD1.sortByKey(false).collect()
```

运行后屏幕显示界面如图 9-27 所示。

```
hduser@master: ~  
scala> kvRDD1.sortByKey(false).collect()  
res11: Array[(Int, Int)] = Array((5,6), (3,4), (3,6), (1,2))
```

图 9-27 sortByKey 从大到小按照 key 排序的范例

步骤 09 reduceByKey

reduceByKey 命令，例如 kvRDD1.reduceByKey((x, y)=>x+y)，会将具有相同 Key 值的数据合并。合并的方式是按照传入的匿名函数(x,y)=>x+y 相加，合并后产生另一个 RDD。以我们的测试数据为例：

1. kvRDD1 数据(3, 4), (3, 6), (5, 6), (1, 2)中，第一个字段是 key、第二个字段是 value。
2. reduceByKey 会寻找相同的 key 合并，测试数据中 key 是 3，有 2 组数据(3, 4), (3, 6)。
3. 经过 reduceByKey 运算后，会将这 2 条数据(3, 4), (3, 6)合并为 1 条(3,10)，合并的方法是将 value 相加 4+6=10。
4. 剩下的(5, 6), (1, 2)，因为没有相同的 key 所以不变。

参考下图（见图 9-28）。

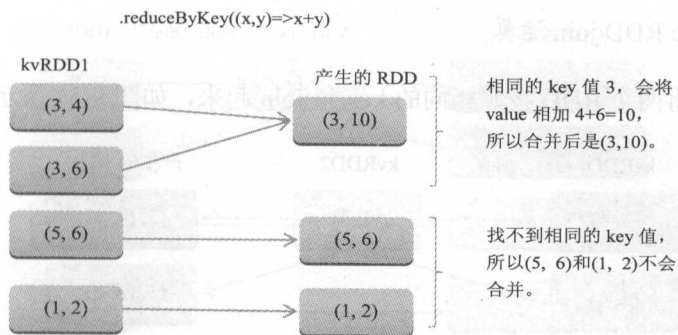


图 9-28 reduceByKey 命令运算范例的图解

➤ 使用匿名函数 reduceByKey 命令

```
kvRDD1.reduceByKey((x,y)=>x+y).collect
```

➤ 使用匿名函数+匿名参数 reduceByKey 命令

```
kvRDD1.reduceByKey(_+_).collect
```

屏幕显示界面如图 9-29 所示, 以上两条语句的运行结果完全相同。

```
hduser@master:~$ scala> kvRDD1.reduceByKey((x,y)=>x+y).collect
res0: Array[(Int, Int)] = Array((1,2), (3,10), (5,6))

scala> kvRDD1.reduceByKey(_+_).collect
res1: Array[(Int, Int)] = Array((1,2), (3,10), (5,6))
```

图 9-29 reduceByKey 两条语句范例的运行结果完全相同

9.6 多个 RDD Key-Value “转换” 运算

步骤 01 Key-Value RDD 范例

为了示范多个 RDD Key-Value “转换” 运算, 先使用下列命令:

➤ 创建范例 Key-Value RDD

```
val kvRDD1 = sc.parallelize(List((3, 4), (3, 6), (5, 6), (1, 2)))
val kvRDD2 = sc.parallelize(List((3, 8)))
```

运行后的屏幕显示界面如图 9-30 所示。

```
hduser@master:~$ scala> val kvRDD1 = sc.parallelize(List((3, 4), (3, 6), (5, 6), (1, 2)))
kvRDD1: org.apache.spark.rdd.RDD[(Int, Int)] = ParallelCollectionRDD[2] at parallelize at <console>:21

scala> val kvRDD2 = sc.parallelize(List((3, 8)))
kvRDD2: org.apache.spark.rdd.RDD[(Int, Int)] = ParallelCollectionRDD[3] at parallelize at <console>:21
```

图 9-30 创建范例 Key-Value RDD

步骤 02 Key-Value RDD join 运算

join 运算可以将两个 RDD 按照相同的 key 值 join 起来, 如图 9-31 所示。

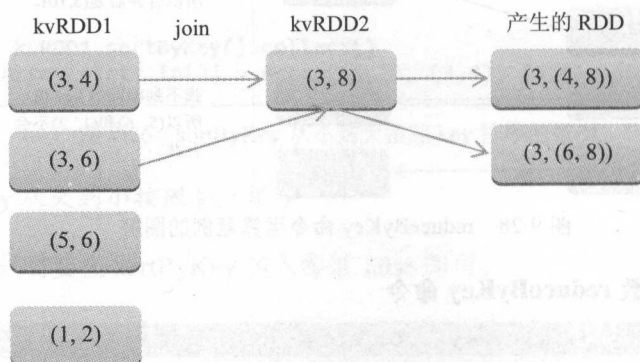


图 9-31 Key-Value RDD join 运算范例的图解

1. kvRDD1 与 kvRDD2 的 key 唯一相同的 key 值是 3。
2. kvRDD1 是(3,4)、(3,6), 而 kvRDD2 是(3,8), 所以 join 的结果如图 9-31 所示。

如下列命令所示, kvRDD1 join kvRDD2 并且输出每一条数据。

```
kvRDD1.join(kvRDD2).foreach(println)
```

运行后, 屏幕显示界面如图 9-32 所示。

```

hduser@master: ~
scala> kvRDD1.join(kvRDD2).foreach(println)
(3,(4,8))
(3,(6,8))
  
```

图 9-32 Key-Value RDD join 运算的范例

步骤 03 Key-Value leftOuterJoin 运算

1. leftOuterJoin 会从左边的集合 (kvRDD1) 对应到右边的集合 (kvRDD2), 并显示所有左边的集合 (kvRDD1) 中的所有元素。
2. 如果 kvRDD1 的 key 值对应到 kvRDD2, 会显示相同的 key (3, (4, Some(8))), (3, (6, Some(8)))。
3. 如果 kvRDD1 的 key 值对应不到 kvRDD2, 就会显示 None (5, (6, None)), (1, (2, None))。

图解如图 9-33 所示。

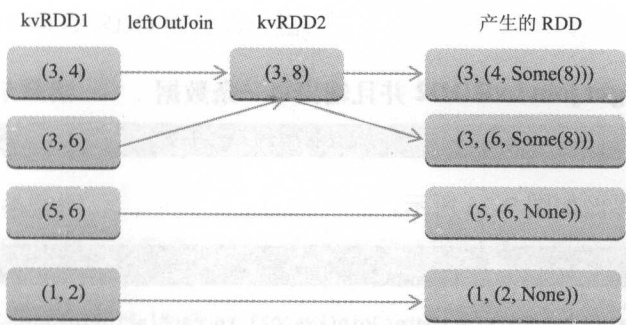


图 9-33 Key-Value leftOuterJoin 运算范例的图解

如下命令所示：

➤ kvRDD1 left join kvRDD2 并且输出每一条数据

```
kvRDD1.leftOuterJoin(kvRDD2).foreach(println)
```

运行后屏幕显示界面如图 9-34 所示。

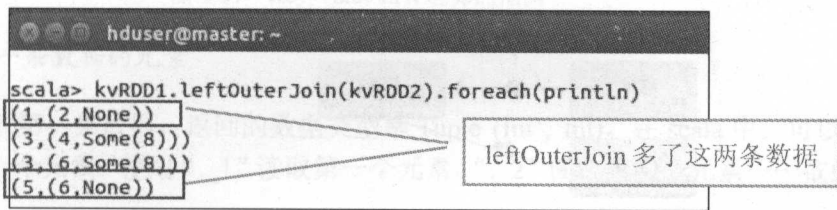


图 9-34 Key-Value leftOuterJoin 运算的范例

步骤 04 Key-Value RDD rightOuterJoin 运算

1. rightOuterJoin 会从右边的集合 (kvRDD2) 对应到左边的集合 (kvRDD1)，并显示所有右边的集合 (kvRDD2) 中的所有元素。
2. 如果 kvRDD2 的 key 值对应到 kvRDD1，会显示重复的 key (3, (Some(4), 8))、(3, (Some(6), 8))。

图解如图 9-35 所示。

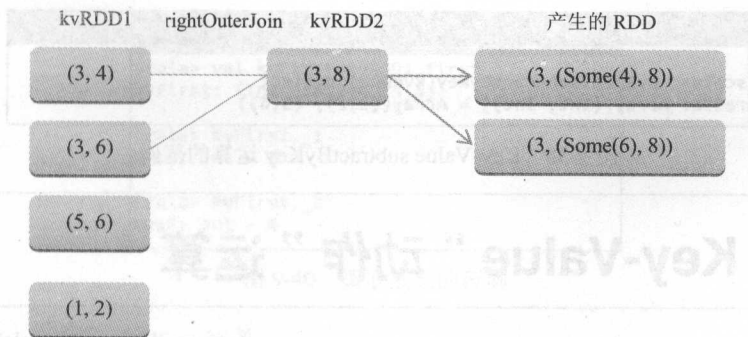


图 9-35 Key-Value RDD rightOuterJoin 运算范例的图解

如下命令所示:

➤ **kvRDD1 right join kvRDD2 并且输出每一条数据**

```
kvRDD1.rightOuterJoin(kvRDD2).foreach(println)
```

运行后屏幕显示界面如图 9-36 所示。

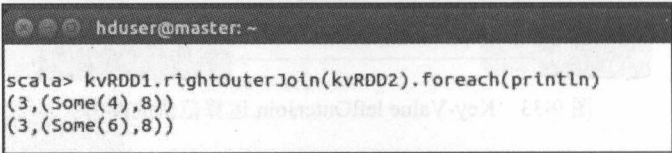


图 9-36 Key-Value RDD rightOuterJoin 运算的范例

步骤 05 Key-Value subtractByKey 运算

subtractByKey 运算会删除相同 key 值的数据。例如 kvRDD1 数据(3, 4), (3, 6), (5, 6), (1, 2), 删除与 kvRDD2 具有相同 key 值 3 的项, 所以只剩下(5, 6)与(1,2), 如图 9-37 所示。

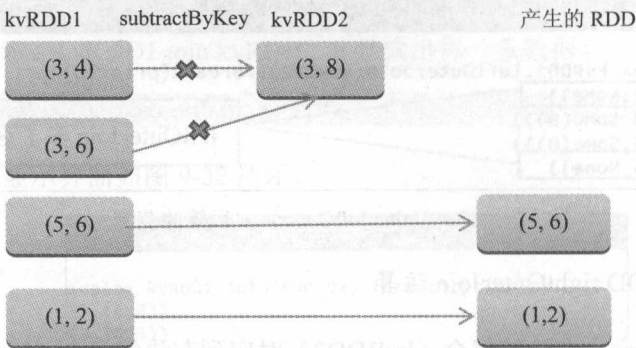


图 9-37 Key-Value subtractByKey 运算范例的图解

➤ **kvRDD1 subtract kvRDD2**

```
kvRDD1.subtractByKey(kvRDD2).collect
```

运行后屏幕显示界面如图 9-38 所示。

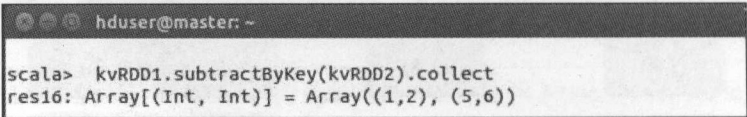


图 9-38 Key-Value subtractByKey 运算的范例

9.7

Key-Value “动作”运算

步骤 01 Key-Value first 运算

可以使用下列命令读取 RDD 部分的数据：

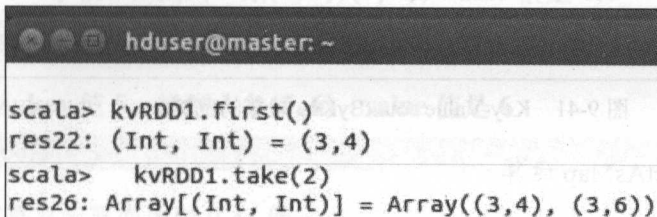
➤ 读取第一条数据

```
kvRDD1.first()
```

➤ 读取前 N 条数据

```
kvRDD1.take(2)
```

运行后屏幕显示界面如图 9-39 所示。



```
hduser@master: ~
scala> kvRDD1.first()
res22: (Int, Int) = (3,4)
scala> kvRDD1.take(2)
res26: Array[(Int, Int)] = Array((3,4), (3,6))
```

图 9-39 Key-Value first 运算的范例

步骤 02 读取第一条数据的元素

使用.first 读取第一条数据，返回的数据类型是 Tuple (Int , Int)。在 scala 中，可以使用这条语句来读取每一个元素：使用“._1”读取第一个元素、“._2”读取第二个元素……依此类推。

➤ 读取第一条数据

```
val kvFirst=kvRDD1.first
```

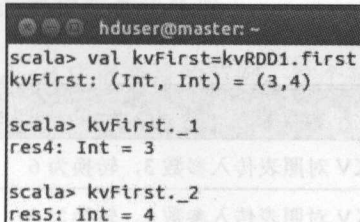
➤ 读取第一条数据的第一个元素，也就是 Key 值

```
kvFirst._1
```

➤ 读取第一笔数据的第二个元素，也就是 Value 值

```
kvFirst._2
```

运行后屏幕显示界面如图 9-40 所示。



```
hduser@master: ~
scala> val kvFirst=kvRDD1.first
kvFirst: (Int, Int) = (3,4)
scala> kvFirst._1
res4: Int = 3
scala> kvFirst._2
res5: Int = 4
```

图 9-40 读取数据的范例

步骤 03 Key-Value countByKey 运算

可以使用下列命令进行运算：

➤ 计算每一个 Key 值的条数

```
kvRDD1.countByKey()
```

例如 kvRDD1 数据(3, 4), (3, 6), (5, 6), (1, 2), key 值为 3 的有两条数据，其余 key 值 1, 5 都只有一条数据。所以结果为：1->1 条、3->2 条、5->1 条。运行后屏幕显示界面如图 9-41 所示。

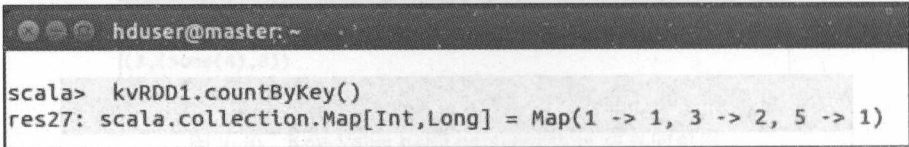


图 9-41 Key-Value countByKey 运算的范例

步骤 04 Key-Value collectAsMap 运算

可以使用 collectAsMap 进行运算：

➤ 创建 Key-Value 的对照表

命令	说明
var KV=kvRDD1.collectAsMap()	创建对照表 KV

例如 kvRDD1 数据(3, 4), (3, 6), (5, 6), (1, 2)能用于产生对照表 Map(5->6, 1->2, 3->6)，不过 Key=3 有两个值 4,6，系统只能自动对应到其中的值 6。

运行后屏幕显示界面如图 9-42 所示。

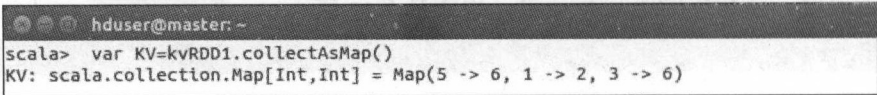


图 9-42 Key-Value collectAsMap 运算的范例

步骤 05 使用对照表转换数据

对照表创建后就可以进行数据转换：

➤ 使用对照表转换数据

命令	说明
KV(3)	KV 对照表传入参数 3，转换为 6
KV(1)	KV 对照表传入参数 1，转换为 2

运行后屏幕显示界面如图 9-43 所示。

```
scala> KV(3)
res31: Int = 6

scala> KV(1)
res32: Int = 2
```

图 9-43 使用对照表转换数据的范例

步骤 06 Key-Value lookup 运算

可以使用 lookup 输入 key 值来查找 value 值。以 kvRDD1 为例((3, 4), (3, 6), (5, 6), (1, 2)):

➤ 输入 key 值 3, 找到 2 条(3, 4), (3, 6), value 值是 4,6

```
kvRDD1.lookup(3)
```

➤ 输入 key 值 5, 找到 1 条(5, 6), value 值是 6

```
kvRDD1.lookup(5)
```

运行后屏幕显示界面如图 9-44 所示。

```
hduser@master: ~
scala> kvRDD1.lookup(3)
res33: Seq[Int] = WrappedArray(4, 6)

scala> kvRDD1.lookup(5)
res34: Seq[Int] = WrappedArray(6)
```

图 9-44 Key-Value lookup 运算的范例

9.8 Broadcast 广播变量

接下来,我们将介绍 Shared variable 共享变量。共享变量可用于节省内存与运行时间,提升并行处理时的执行效率。共享变量包括:

- Broadcast 广播变量——本节会介绍
- accumulator 累加器——下一节会介绍

步骤 01 不使用 Broadcast 广播变量的范例

我们先看不使用 Broadcast 广播变量的范例。这个范例很简单,说明如下:
先创建水果编号与名称对照表,然后使用此对照表,将水果编号转换为水果名称。

➤ 先创建 kvFruit

这是水果编号与名称的 Key-Value RDD。

```
val kvFruit=sc.parallelize(List((1,"apple"), (2,"orange"), (3,"banana"),
(4,"grape")))
```

➤ 创建 fruitMap 对照表

使用 collectAsMap 创建 fruitMap（水果编号与名称对照表）。

```
val fruitMap=kvFruit.collectAsMap()
```

➤ 创建 fruitIds

这是只有水果编号的 RDD。

```
val fruitIds=sc.parallelize(List(2,4,1,3))
```

➤ 使用 fruitMap 对照表进行转换

将每一个 fruitIds 水果编号，使用 fruitMap 对照表转换为 fruitNames 水果名称。

```
val fruitNames= fruitIds.map(x=>fruitMap(x)).collect
```

运行后屏幕显示界面如图 9-45 所示。

```
hduser@master: ~
scala> val kvFruit = sc.parallelize(List((1, "apple"), (2, "orange"), (3, "banana"),
(4, "grape")))
kvFruit: org.apache.spark.rdd.RDD[(Int, String)] = ParallelCollectionRDD[2] at parallelize at <console>:21

scala> val fruitMap=kvFruit.collectAsMap()
fruitMap: scala.collection.Map[Int,String] = Map(2 -> orange, 4 -> grape, 1 -> apple, 3 -> banana)

scala> val fruitIds=sc.parallelize(List(2,4,1,3))
fruitIds: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[3] at parallelize at <console>:21

scala> val fruitNames= fruitIds.map(x=>fruitMap(x)).collect
fruitNames: Array[String] = Array(orange, grape, apple, banana)
```

先创建 kvFruit

创建 fruitMap

创建 fruitIds

将 fruitIds 水果编号转换为 fruitNames 水果名称

图 9-45 不使用 Broadcast 广播变量的范例

以上的范例执行起来没有任何问题。但是在并行处理中，每执行一次转换，都必须将 fruitIds 与 fruitMap 传送到 Worker Node,才能够执行转换,如图 9-46 所示。如果对照表 fruitMap 很大，而且需要转换的 fruitIds 水果编号 RDD 也很大，则会耗费很多内存与时间。

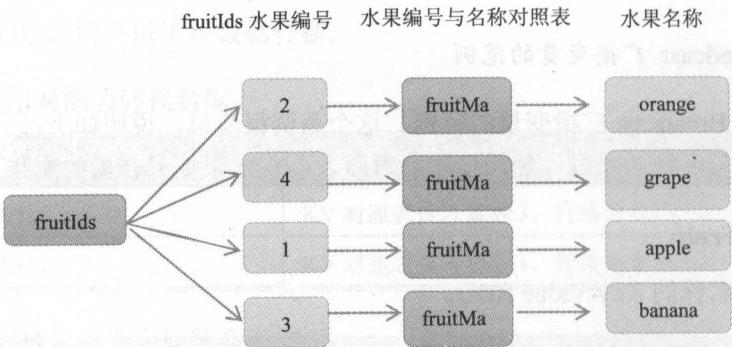


图 9-46 将水果编号转换为水果名称范例的示意图

为了解决这个问题，就必须使用 Broadcast 广播变量：

步骤 02 使用 Broadcast 广播变量的范例

Broadcast 广播变量使用规则如下：

1. 可以使用 `SparkContext.broadcast([初始值])` 创建；
2. 使用 `.value` 的方法来读取广播变量的值；
3. Broadcast 广播变量被创建后，就不可以修改。

下列使用 Broadcast 广播变量的范例与之前的范例类似，不同之处是使用 `sc.broadcast` 传入 `fruitMap` 作为参数，创建 `bcFruitMap` 广播变量。使用 `bcFruitMap.value(x)` 广播变量，转换为 `fruitNames` 水果名称。

下列程序代码与之前如图 9-45 所示的程序代码类似，不同之处是改用了 `bcFruitMap` 广播变量：

➤ 创建 kvFruit

这是水果编号与名称的 Key-Value RDD。

```
val kvFruit=sc.parallelize(List((1,"apple"), (2,"orange"), (3,"banana"),
(4,"grape")))
```

➤ 创建 fruitMap 对照表

使用 `collectAsMap` 创建 `fruitMap`（水果编号与名称对照表）。

```
val fruitMap=kvFruit.collectAsMap()
```

➤ 创建 bcFruitMap 广播变量

使用 `sc.broadcast` 传入 `fruitMap` 参数，创建 `bcFruitMap` 广播变量。

```
val bcFruitMap=sc.broadcast(fruitMap)
```

➤ 再创建 fruitIds

这是只有水果编号的 RDD。

```
val fruitIds=sc.parallelize(List(2,4,1,3))
```

➤ 使用 bcFruitMap.value(x) 广播变量进行转换

将 `fruitIds` 水果编号，使用 `bcFruitMap.value(x)` 广播变量转换为 `fruitNames` 水果名称。

```
val fruitNames= fruitIds.map(x=>bcFruitMap.value(x)).collect
```

运行后屏幕显示界面如图 9-47 所示，与之前的结果相同。

```
scala> val kvFruit = sc.parallelize(List((1, "apple"), (2, "orange"), (3, "banana"),
(4, "grape")))
kvFruit: org.apache.spark.rdd.RDD[(Int, String)] = ParallelCollectionRDD[5] at parallelize at <console>:21

scala> val fruitMap=kvFruit.collectAsMap()
fruitMap: scala.collection.Map[Int,String] = Map(2 -> orange, 4 -> grape, 1 -> apple, 3 -> banana)

scala> val bcFruitMap=sc.broadcast(fruitMap)
bcFruitMap: org.apache.spark.broadcast.Broadcast[scala.collection.Map[Int,String]] = Broadcast(3)

scala> val fruitIds=sc.parallelize(List(2,4,1,3))
fruitIds: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[6] at parallelize at <console>:21

scala> val fruitNames= fruitIds.map(x=>bcFruitMap.value(x)).collect()
fruitNames: Array[String] = Array(orange, grape, apple, banana)
```

使用 `sc.broadcast` 传入 `fruitMap` 参数，创建 `bcFruitMap` 广播变量

使用 `bcFruitMap.value(x)` 广播变量，将 `fruitIds` 转换为 `fruitNames`

图 9-47 使用 Broadcast 广播变量的范例

执行的结果与之前步骤 1 范例相同。如图 9-48 所示，在并行处理中，`bcFruitMap` 广播变量会传送到 Worker Node 机器，并且存储在内存中。后续在此 Worker Node 都可以使用此 `bcFruitMap` 广播变量执行转换，这样就可以节省很多内存与传送时间。

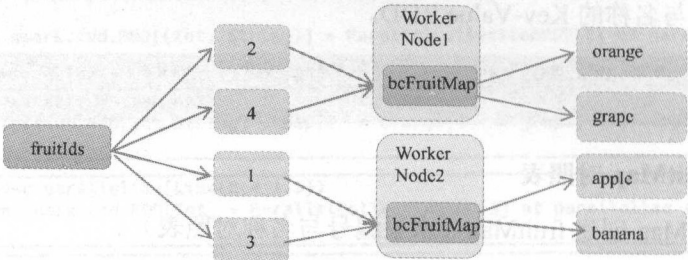


图 9-48 使用 Broadcast 广播变量执行范例的图解

9.9 accumulator 累加器

步骤 01 accumulator 累加器的介绍

加总是 MapReduce 常用的运算，然而为了方便并行处理，Spark 特别提供了 accumulator 累加器共享变量（Shared variable）。使用规则如下：

- accumulator 累加器可以使用 `SparkContext.accumulator([初始值])` 来创建。
- 使用 “+=” 累加。
- 在 task 中，例如 foreach 循环中，不能读取累加器的值。
- 只有驱动程序，也就是循环外，才可以使用 `.value` 来读取累加器的值。

步骤 02 accumulator 累加器范例

接下来，我们使用下列范例计算 RDD 的求和、计数：

➤ 创建范例 RDD

```
val intRDD = sc.parallelize(List(3,1, 2, 5, 5))
```

➤ 创建 total 累加器

此处的初始值使用 0.0，所以是 Double 的类型。

```
val total = sc.accumulator(0.0)
```

➤ 创建 num 累加器

此处的初始值使用 0，所以是 Int 的类型。

```
val num = sc.accumulator(0)
```

➤ 使用 foreach 传入参数 i，针对每一条数据执行

total 累加 intRDD 元素的值、num 累加 intRDD 元素的数量。

```
intRDD.foreach(i => {
  total += i
  num += 1
})
```

➤ 显示求和、计数

```
println("total="+total.value+", num="+num.value)
```

➤ 计算平均=求和 / 计数

```
val avg = total.value / num.value
```

运行后屏幕显示界面如图 9-49 所示。

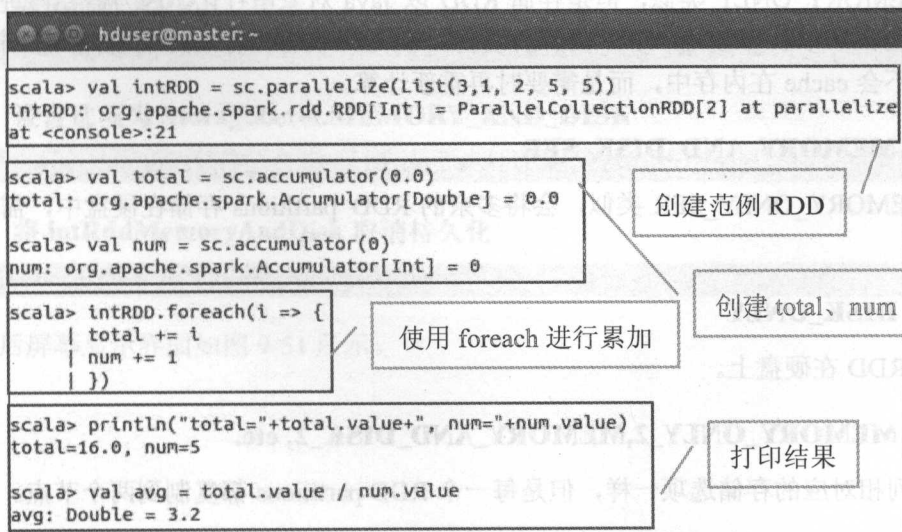


图 9-49 accumulator 累加器的范例

9.10 RDD Persistence 持久化

步骤 01 RDD Persistence 持久化介绍

Spark RDD 持久化机制，可以用于将需要重复运算的 RDD 存储在内存中，以便大幅提升运算效率。

Spark RDD 持久化使用方法如下：

- `RDD.persist()`（存储等级）——可以指定存储等级，默认是 `MEMORY_ONLY`，也就是存储在内存中。
- `RDD.unpersist()`——取消持久化。

持久化存储等级说明如下：

➤ `MEMORY_ONLY`

这是默认选项。

存储 RDD 的方式是以 Java 对象反串行化在 JVM 内存中。如果 RDD 太大无法完全存储在内存，多余的 RDD partitions 不会 cache 在内存中，而是需要时再重新计算。

➤ `MEMORY_AND_DISK`

存储 RDD 的方式是以 Java 对象反串行化在 JVM 内存中。如果 RDD 太大无法完全存储在内存。会将多余的 RDD partitions 存储在硬盘中，需要时再从硬盘读取。

➤ `MEMORY_ONLY_SER`

与 `MEMORY_ONLY` 类似，但是存储 RDD 以 Java 对象串行化，因为需要再进行反串行化才能使用，所以会多使用 CPU 的计算资源，但是比较省内存的存储空间。多余的 RDD partitions 不会 cache 在内存中，而是需要时再重新计算。

➤ `MEMORY_AND_DISK_SER`

与 `MEMORY_ONLY_SER` 类似，会将多余的 RDD partitions 存储在硬盘中，需要时再从硬盘读取。

➤ `DISK_ONLY`

存储 RDD 在硬盘上。

➤ `MEMORY_ONLY_2, MEMORY_AND_DISK_2, etc.`

与上列相对应的存储选项一样，但是每一个 RDD partitions 都复制到两个节点。

步骤 02 `RDD.persist()` 范例

RDD.persist 持久化范例如下：

➤ 创建范例 RDD

```
val intRddMemory = sc.parallelize(List(3,1, 2, 5, 5))
```

➤ 将 intRddMemory 持久化

```
intRddMemory.persist()
```

➤ 将 intRddMemory 取消持久化

```
intRddMemory.unpersist()
```

运行后屏幕显示界面如图 9-50 所示。

```
hduser@master: -
scala> val intRddMemory = sc.parallelize(List(3,1, 2, 5, 5))
intRddMemory: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[13] at parallelize at <console>:28

scala> intRddMemory.persist()
res41: intRddMemory.type = ParallelCollectionRDD[13] at parallelize at <console>:28

scala> intRddMemory.unpersist()
res42: intRddMemory.type = ParallelCollectionRDD[13] at parallelize at <console>:28
```

图 9-50 RDD.persist()范例

步骤 03 RDD.persist 设置存储等级范例

RDD.persist 设置存储等级范例如下：

➤ 设置存储等级，必须先导入相关链接库

```
import org.apache.spark.storage.StorageLevel
```

➤ 创建范例 RDD

```
val intRddMemoryAndDisk = sc.parallelize(List(3,1, 2, 5, 5))
```

➤ 设置选项为 StorageLevel.MEMORY_AND_DISK

```
intRddMemoryAndDisk.persist(StorageLevel.MEMORY_AND_DISK)
```

➤ 将 intRddMemoryAndDisk 取消持久化

```
intRddMemoryAndDisk.unpersist()
```

运行后屏幕显示界面如图 9-51 所示。

```

hduser@master:~
scala> import org.apache.spark.storage.StorageLevel
import org.apache.spark.storage.StorageLevel

scala> val intRddMemoryAndDisk = sc.parallelize(List(3,1, 2, 5, 5))
intRddMemoryAndDisk: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[14] at parallelize at <console>:30

scala>

scala> intRddMemoryAndDisk.persist(StorageLevel.MEMORY_AND_DISK)
res43: intRddMemoryAndDisk.type = ParallelCollectionRDD[14] at parallelize at <console>:30

scala> intRddMemoryAndDisk.unpersist()
res44: intRddMemoryAndDisk.type = ParallelCollectionRDD[14] at parallelize at <console>:30

```

图 9-51 RDD.persist 设置存储等级范例

9.11 使用 Spark 创建 WordCount

本章到目前为止，我们学到了 map 与 reduceByKey。接下来，我们将使用 map 与 reduceByKey 编写 Spark 版本的 WordCount。

步骤 01 创建测试文件

在“终端”程序的提示符后输入下列命令：

➤ 创建 WordCount 的数据目录

```
mkdir -p ~/workspace/WordCount/data
```

➤ 切换至 WordCount 的数据目录

```
cd ~/workspace/WordCount/data
```

➤ 编辑 test.txt

```
gedit test.txt
```

运行后屏幕显示界面如图 9-52 所示。

```

hduser@master: ~/workspace/WordCount/data
hduser@master:~$ cd ~/workspace/WordCount/data
hduser@master:~/workspace/WordCount/data$ gedit test.txt

```

图 9-52 创建测试文件

输入后按 Enter 键就会打开 test.txt，屏幕显示界面如图 9-53 所示。

```

*test.txt (~) - gedit
[Icons] 打开 保存
*test.txt x
Apple Apple Orange
Banana Grape Grape

```

输入下列内容

图 9-53 打开 test.txt 进行编制

步骤 02 进入 spark-shell

在~/workspace/WordCount/data 目录下输入 spark-shell，进入 spark 交互界面，如图 9-54 所示。

```
hdbuser@master: ~/workspace/WordCount/data$ spark-shell
15/07/05 10:55:50 WARN NativeCodeLoader: Unable to load native-hadoop library for
your platform... using builtin-java classes where applicable
Welcome to

  ____  _
 / ___|| | | |
| |___| |_| |
 \___|_____|_|

 version 1.4.0

Using Scala version 2.10.4 (OpenJDK 64-Bit Server VM, Java 1.7.0_79)
Type in expressions to have them evaluated.
Spark context available as sc.
15/07/05 10:56:01 WARN Connection: BoneCP specified but not present in CLASSPATH
(or one of dependencies)
15/07/05 10:56:01 WARN Connection: BoneCP specified but not present in CLASSPATH
(or one of dependencies)
15/07/05 10:56:07 WARN ObjectStore: Version information not found in metastore.
hive.metastore.schema.verification is not enabled so recording the schema versio
n 0.13.1aa
SQL context available as sqlContext.

scala>
```

图 9-54 进入 spark-shell 交互界面

步骤 03 执行 WordCount spark 命令

在 Spark 运行 WordCount 就是如此简单，只需要 4 行命令。与第 7 章 Hadoop MapReduce 介绍的 wordCount.java 比较，可以发现程序代码简单很多，而且容易理解。在 spark-shell 输入下列命令，这些命令的详细用法，我们下一节会详细解说：

- ## ➤ 读取文本文件

```
val textFile=sc.textFile("file:/home/hduser/workspace/WordCount/data/test.txt")
```

- 使用 **flatMap** 空格符分隔单词，并读取每个单词

```
val stringRDD=textFile.flatMap(line => line.split(" "))
```

- 通过 map reduce 计算每一个单词出现的次数

```
val countsRDD=stringRDD.map(word => (word, 1)).reduceByKey( + )
```

- ### ► 保存计算结果

```
countsRDD.saveAsTextFile("file:/home/hduser/workspace/WordCount/data/output")
}
```

- ## ➤ 退出 spark-shell

exit

运行后屏幕显示界面如图 9-55 所示。

```
hduser@master: ~/workspace/WordCount/data
scala> val textFile = sc.textFile ( "file:/home/hduser/workspace/WordCount/data/test.txt" )
textFile: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[1] at textFile at <console>:21
scala> val stringRDD=textFile.flatMap(line => line.split(" "))
stringRDD: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[2] at flatMap at <console>:23
scala> val countsRDD = stringRDD.map(word =>
  | (word, 1)).reduceByKey(_+_ )
countsRDD: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[4] at reduceByKey at <console>:26
scala> countsRDD.saveAsTextFile("file:/home/hduser/workspace/WordCount/data/output")
scala> exit
```

图 9-55 执行 WordCount spark 命令

步骤 04 查看 data 目录

exit 回到“终端”程序提示符下，使用 ll 查看 data 目录。

```
ll
```

如图 9-56 所示，saveAsTextFile 会将输出数据保存在 output 目录。

```
hduser@master: ~/workspace/WordCount/data
hduser@master:~/workspace/WordCount/data$ ll
总用量 20
drwxrwxr-x 3 hduser hduser 4096 5月 16 15:43 ./
-rw-rw-r-- 1 hduser hduser 703 5月 16 15:44 derby.log
drwxrwxr-x 2 hduser hduser 4096 5月 16 15:45 output/
-rw-rw-r-- 1 hduser hduser 38 5月 16 15:43 test.txt
hduser@master:~/workspace/WordCount/data$
```

图 9-56 查看 data 目录

步骤 05 查看 output 目录

在终端程序的提示符后面，输入下列命令切换至 output 目录，查看输出目录。

```
cd output
ll
```

运行后屏幕显示界面如图 9-57 所示。

```
hduser@master: ~/workspace/WordCount/data/output
hduser@master:~/workspace/WordCount/data/output$ ll
总用量 20
drwxrwxr-x 2 hduser hduser 4096 5月 16 15:45 ./
drwxrwxr-x 3 hduser hduser 4096 5月 16 15:45 ../
-rw-r--r-- 1 hduser hduser 42 5月 16 15:45 part-00000
-rw-rw-r-- 1 hduser hduser 12 5月 16 15:45 part-00000.crc
-rw-r--r-- 1 hduser hduser 0 5月 16 15:45 _SUCCESS
-rw-rw-r-- 1 hduser hduser 8 5月 16 15:45 _SUCCESS.crc
hduser@master:~/workspace/WordCount/data/output$
```

图 9-57 查看 output 目录

步骤 06 查看 part-00000 输出文件

在“终端”程序提示符后输入下列命令，查看输出文件内容：

```
cat part-00000
```

运行后屏幕显示界面如图 9-58 所示。

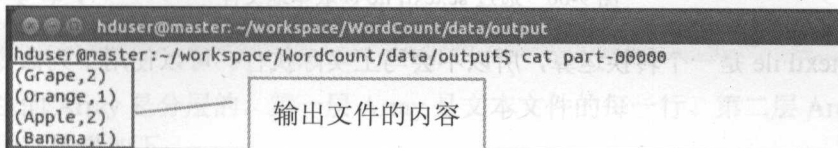


图 9-58 查看 part-00000 输出文件

可以看到 Grape 与 Apple 都是出现了两次，而 Orange 与 Banana 都只有出现一次。

9.12 Spark WordCount 详细解说

上一节已经示范了 Spark 的 WordCount 程序，在本节将详细介绍 WordCount 每一个命令。示意图如图 9-59 所示。

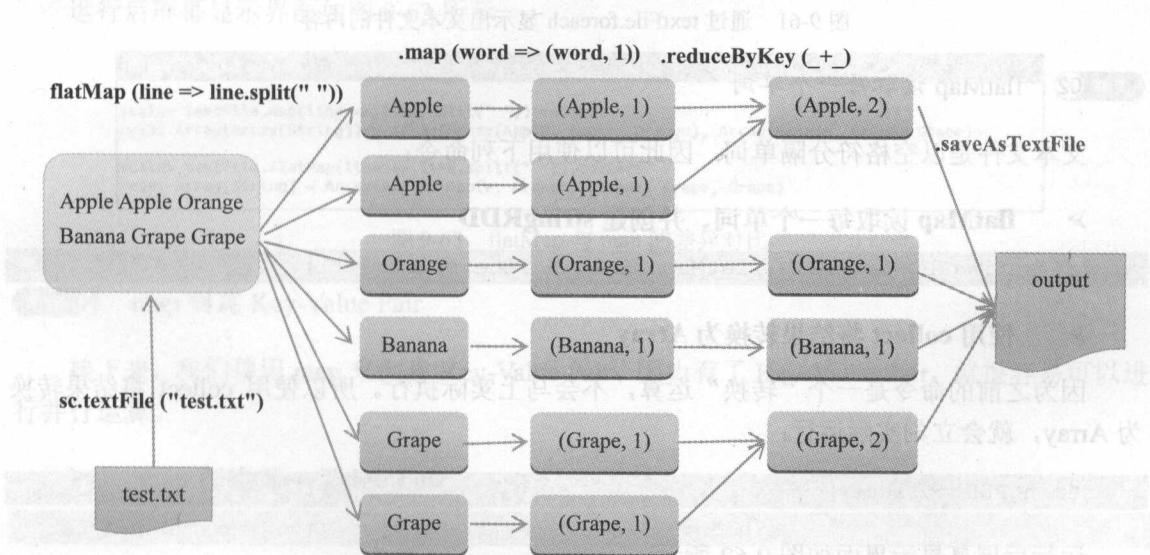


图 9-59 Spark WordCount 运行流程示意图

步骤 01 sc.textFile 读取本地文件。

首先，输入下列命令读取本地文本文件。

```
val textFile = sc.textFile("file:/home/hduser/workspace/WordCount/data/test.txt")
```

运行后屏幕显示界面如图 9-60 所示。

```
hduser@master: ~
scala> val textFile = sc.textFile("file:/home/hduser/workspace/WordCount/data/test.txt")
textFile: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[11] at textFile at <console>:21
```

图 9-60 通过 sc.textFile 读取本地文件

因为 sc.textFile 是一个转换运算，所以不会马上实际执行。可以使用下列命令，显示出文本文件内容：

➤ 显示出文本文件的内容

```
textFile.foreach(println)
```

运行后屏幕显示界面如图 9-61 所示。

```
hduser@master: ~/workspace/WordCount/data
scala> textFile.foreach(println)
Apple Apple Orange
Banana Grape Grape
```

图 9-61 通过 textFile.foreach 显示出文本文件的内容

步骤 02 flatMap 读取每一个单词

文本文件是以空格符分隔单词，因此可以使用下列命令：

➤ flatMap 读取每一个单词，并创建 stringRDD

```
val stringRDD=textFile.flatMap(line => line.split(" "))
```

➤ 使用 collect 将结果转换为 Array

因为之前的命令是一个“转换”运算，不会马上实际执行。所以使用 collect 将结果转换为 Array，就会立刻实际运行：

```
stringRDD.collect
```

运行后屏幕显示界面如图 9-62 所示。

```
hduser@master: ~/workspace/WordCount/data
scala> val stringRDD=textFile.flatMap(line => line.split(" "))
stringRDD: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[7] at flatMap at <console>:23

scala> stringRDD.collect
res7: Array[String] = Array(Apple, Apple, Orange, Banana, Grape, Grape)
```

图 9-62 flatMap 读取单词的范例

步骤 03 flatMap 与 map 的差异

在这里要特别注意,在上一步骤以空格符分隔单词创建 stringRDD 时,必须要使用 flatMap 不能使用 map, 因为 flatMap 与 map 功能不同。你可以测试下列两个命令, 就可以了解这两个命令的差异:

➤ map 命令

```
textFile.map(line => line.split(" ")).collect
```

map 产生的 Array 是分层的。第一层 Array 是文本文件的每一行、第二层 Array 是每一行内的英文单词。结果如下:

```
Array(Array(Apple, Apple, Orange), Array(Banana, Grape, Grape))
```

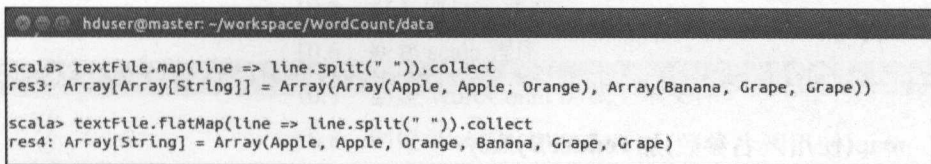
➤ flatMap 命令

```
textFile.flatMap(line => line.split(" ")).collect
```

flat 有平坦的意思, 也就是说 flatMap 产生的 Array, 会将所有分层去掉, 结果如下:

```
Array(Apple, Apple, Orange, Banana, Grape, Grape)
```

运行后屏幕显示界面如图 9-63 所示。



```
scala> textFile.map(line => line.split(" ")).collect
res3: Array[Array[String]] = Array(Array(Apple, Apple, Orange), Array(Banana, Grape, Grape))

scala> textFile.flatMap(line => line.split(" ")).collect
res4: Array[String] = Array(Apple, Apple, Orange, Banana, Grape, Grape)
```

图 9-63 flatMap 与 map 的差异对比

步骤 04 map 创建 Key-Value Pair

接下来, 我们使用 map 来创建 Key-Value Pair。因为有了 Key-Value Pair, 就很容易可以进行并行运算。

➤ map 创建 Key-Value Pair

```
stringRDD.map(word => (word, 1)).foreach(println)
```

1. 使用 map 命令将 stringRDD 每一个英文单词转换为 Key-Value。
2. 其中 Key 值是每一个英文单词、Value 值都是 1。
3. 因为 map 是“转换”运算, 所以不会马上实际运行。为了立刻运行看到结果, 可以加上“行动”运算.foreach(println)输出每一条数据。

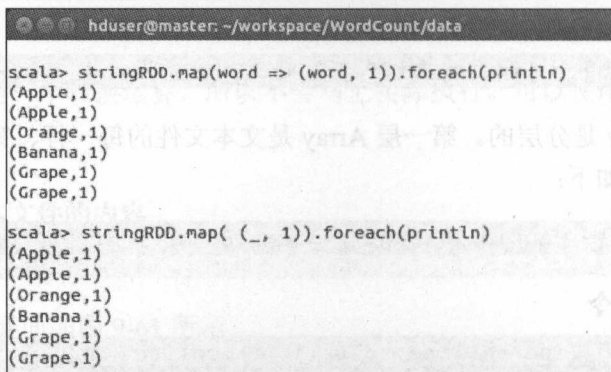
➤ map 匿名参数创建 Key-Value Pair

在 scala 语言中, 可以使用下划线符号“_”将“word => (word)”省略。如使用下列命令

更简洁:

```
stringRDD.map( (_, 1)).foreach(println)
```

这两个命令的运行结果完全相同，运行结果的屏幕显示界面如图 9-64 所示，每一个单词的 value 都是 1:



```

hduser@master: ~/workspace/WordCount/data
scala> stringRDD.map(word => (word, 1)).foreach(println)
(Apple,1)
(Apple,1)
(Orange,1)
(Banana,1)
(Grape,1)
(Grape,1)

scala> stringRDD.map( (_, 1)).foreach(println)
(Apple,1)
(Apple,1)
(Orange,1)
(Banana,1)
(Grape,1)
(Grape,1)

```

图 9-64 map 创建 Key-Value Pair 的两种命令，运行结果完全相同

步骤 05 map 加 reduceByKey

接下来，使用 map 加上 reduceByKey 完成 Map/Reduce 功能。

➤ map 加 reduceByKey

```
stringRDD.map(word => (word, 1)).reduceByKey((x,y)=>(x+y)).foreach(println)
```

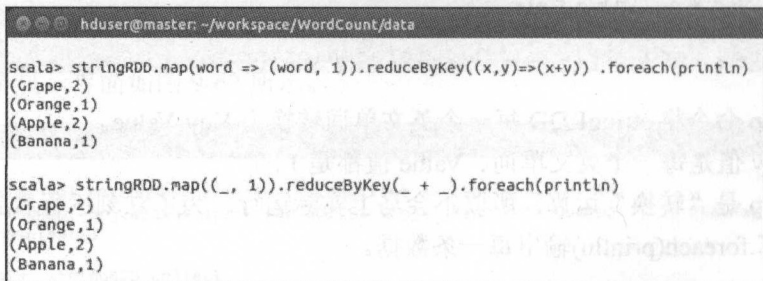
➤ map(使用匿名参数)加 reduceByKey

```
stringRDD.map( (_, 1)).reduceByKey(_ + _).foreach(println)
```

reduceByKey 传入匿名函数(x,y)=>(x+y)，此匿名函数会将相同的 key 值相加。

因为 reduceByKey 是“转换”运算，所以不会马上运行。你可以加上“行动”运算.foreach(println)输出每一条数据。

以上两个命令运行结果完全相同。运行结果的屏幕显示界面如图 9-65 所示。



```

hduser@master: ~/workspace/WordCount/data
scala> stringRDD.map(word => (word, 1)).reduceByKey((x,y)=>(x+y)).foreach(println)
(Grape,2)
(Orange,1)
(Apple,2)
(Banana,1)

scala> stringRDD.map( (_, 1)).reduceByKey(_ + _).foreach(println)
(Grape,2)
(Orange,1)
(Apple,2)
(Banana,1)

```

图 9-65 使用 map 加上 reduceByKey 完成 Map/Reduce 功能

可以看到 Grape 与 Apple 都是出现两次，而 Orange 与 Banana 都只有出现一次。

第 10 章

Spark的集成开发环境

- 10.1 下载与安装 eclipse Scala IDE
- 10.2 下载项目所需要的 Library
- 10.3 启动 eclipse
- 10.4 创建新的 Spark 项目
- 10.5 设置项目链接库
- 10.6 新建 scala 程序
- 10.7 创建 WordCount 测试文本文件
- 10.8 创建 WordCount.scala
- 10.9 编译 WordCount.scala 程序
- 10.10 运行 WordCount.scala 程序
- 10.11 导出 jar 文件
- 10.12 spark-submit 的详细介绍
- 10.13 在本地 local 模式运行 WordCount 程序
- 10.14 在 Hadoop yarn-client 运行 WordCount 程序
- 10.15 在 Spark Standalone Cluster 上运行 WordCount 程序
- 10.16 本书范例程序的安装说明

之前我们介绍 spark-shell 的优点是具有交互性，输入命令后可以立刻看到响应。然而，缺点是无法重复使用。如果我们希望执行某一功能，所有命令必须再执行一次。

本章我们将介绍使用 eclipse 集成开发环境 (IDE)，来开发 Spark 应用程序。eclipse 是很受欢迎的跨平台、开放源码、集成开发环境 (IDE)。eclipse 是一个开发框架，最初用于开发 Java 语言的程序，并且可以通过外挂模块的支持，使其称为其他程序设计语言的开发工具，例如 C++、Python、PHP、Scala 等。

使用集成开发环境的好处很多，举例如下：

1. Code Completion: 自动完成程序代码。如图 10-1 所示，当输入 sc 然后按下“.”就会弹出此对象能够使用的 method。有了这个功能，就不需要强记命令了，非常方便。

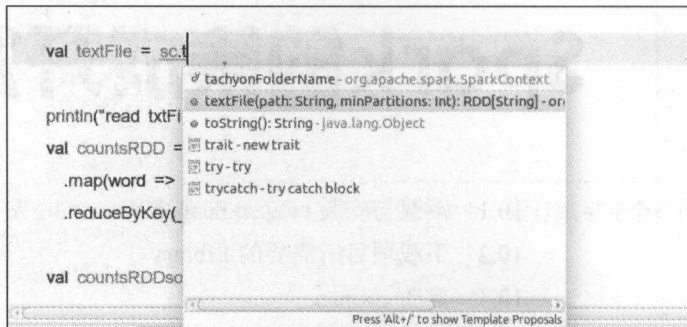


图 10-1 自动完成程序代码的功能

2. Semantic Highlighting: 关键词、对象、变量、字符串、注释……都用不同的颜色标注出来，让程序代码更易读，如图 10-2 所示。

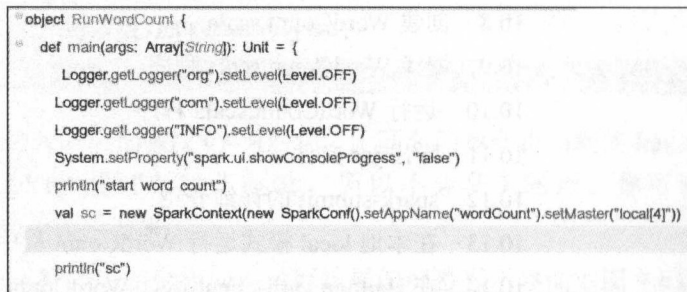


图 10-2 自动完成程序代码的功能

3. 程序代码的编辑、编译、调试、运行都在同一个环境中，大幅度提升了软件开发的效率。如图 10-3 所示。

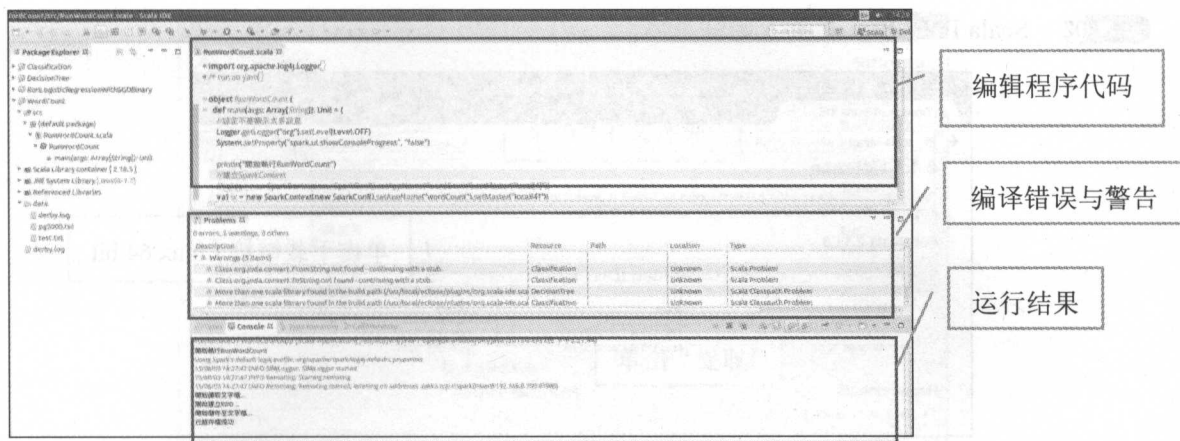


图 10-3 集成开发环境

本章命令整理

本章使用的命令已经整理在本书的博客文章中，在练习安装时可以复制博客文章中的命令，然后粘贴到“终端”程序中。这样可节省你打字的时间，也不用担心打错字（如果你无法在 VirtualBox 虚拟机的 Ubuntu “终端”程序中执行复制/粘贴操作，参考第 3.9 节的说明，设置好 VirtualBox 的共享剪贴板）。本书的博客网址为：

<http://blog.sina.com.cn/hadoopsparkbook>

10.1 下载与安装 eclipse Scala IDE

首先下载 Scala IDE eclipse 集成开发环境，我们将安装在 master 服务器上。

步骤 01 浏览 Scala IDE 网页

在 master 服务器中，启动浏览器输入下列网址，进入 Scala IDE 官网，如图 10-4 所示。

<http://scala-ide.org/>



图 10-4 浏览 Scala IDE 网页

步骤 02 Scala IDE 下载页面（见图 10-5）

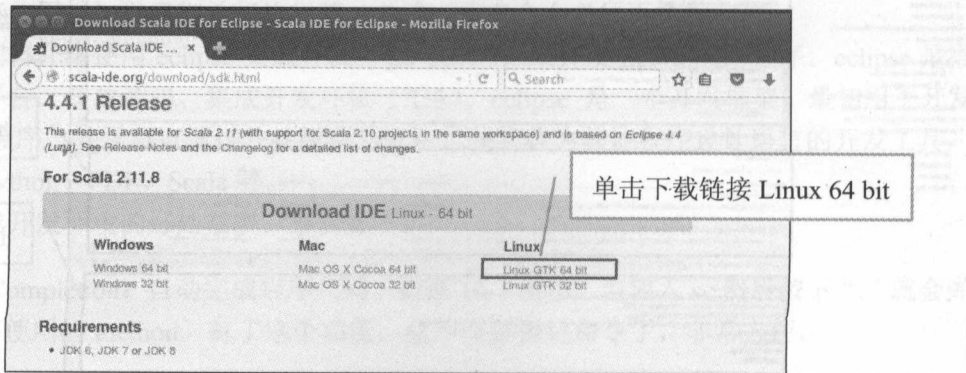


图 10-5 Scala IDE 下载页面

步骤 03 选择下载后，选择以“归档管理器”来打开，如图 10-6 所示。

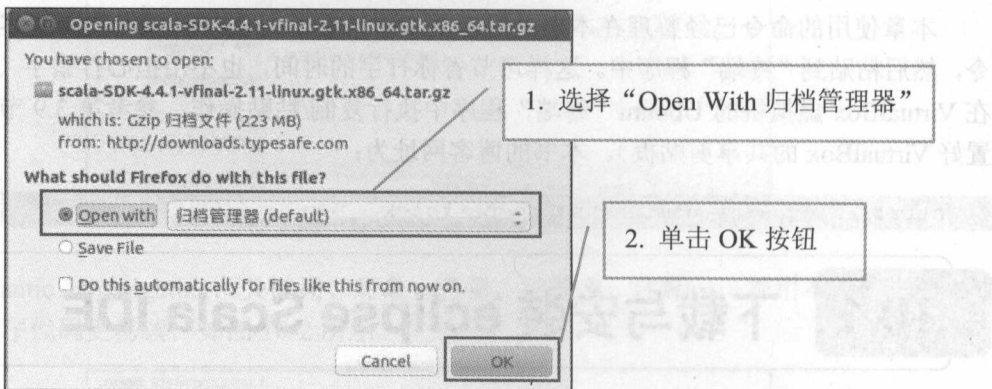


图 10-6 选择“Open With 归档管理器”

步骤 04 归档管理器界面

在归档管理器中解压缩文件，如图 10-7 所示。

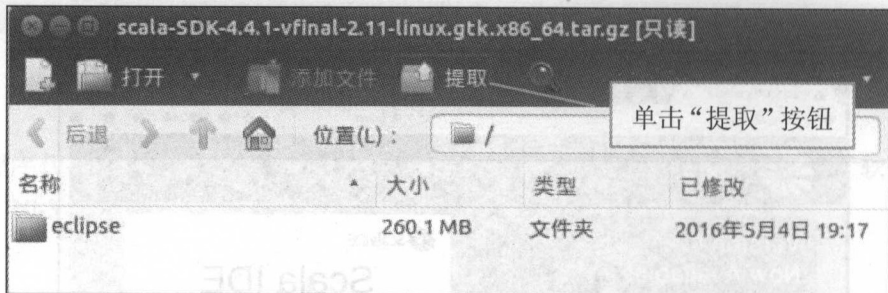


图 10-7 在归档管理器中“提取”压缩文件

步骤 05 选择解压缩的文件夹

接下来选择解压缩的文件夹，解压缩到“主文件夹”，屏幕显示界面如图 10-8 所示。

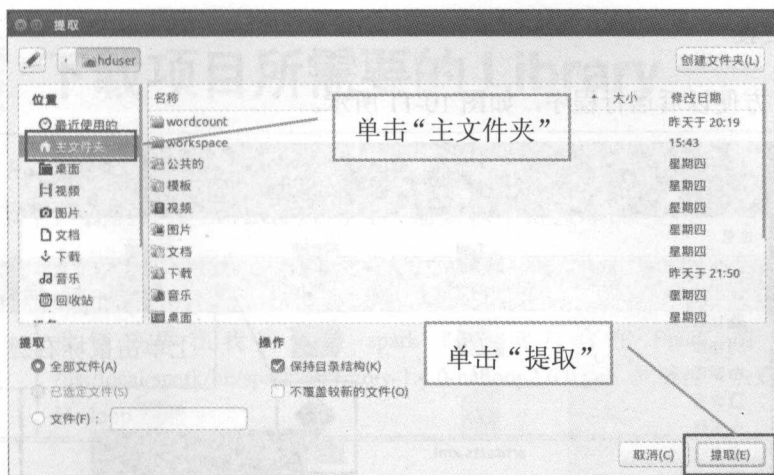


图 10-8 选择解压缩文件夹

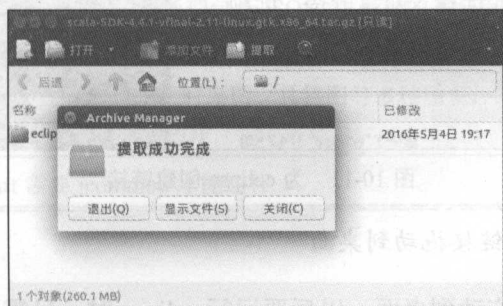
步骤 06 解压缩成功 (见图 10-9)

图 10-9 解压缩成功的屏幕显示界面

步骤 07 启动文件资源管理器

启动文件资源管理器，然后打开 eclipse 的文件夹，如图 10-10 所示。

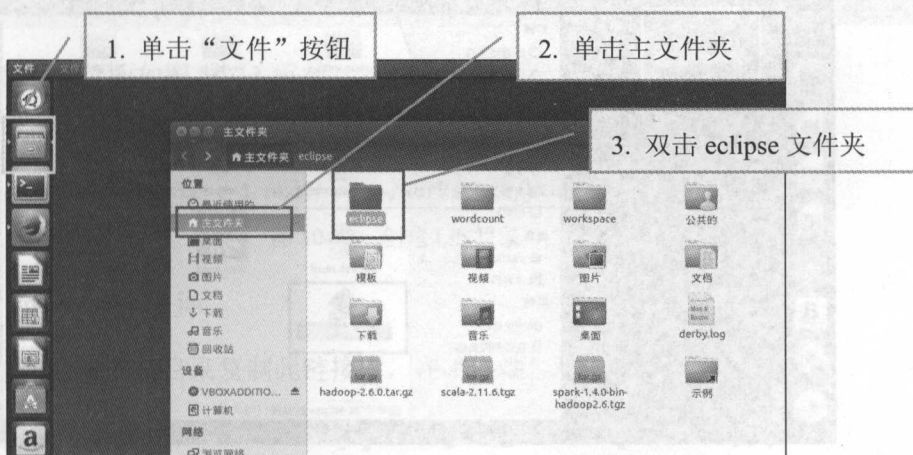


图 10-10 启动文件资源管理器，然后打开 eclipse 的文件夹

步骤 08 创建链接

创建链接，方便以后运行程序，如图 10-11 所示。

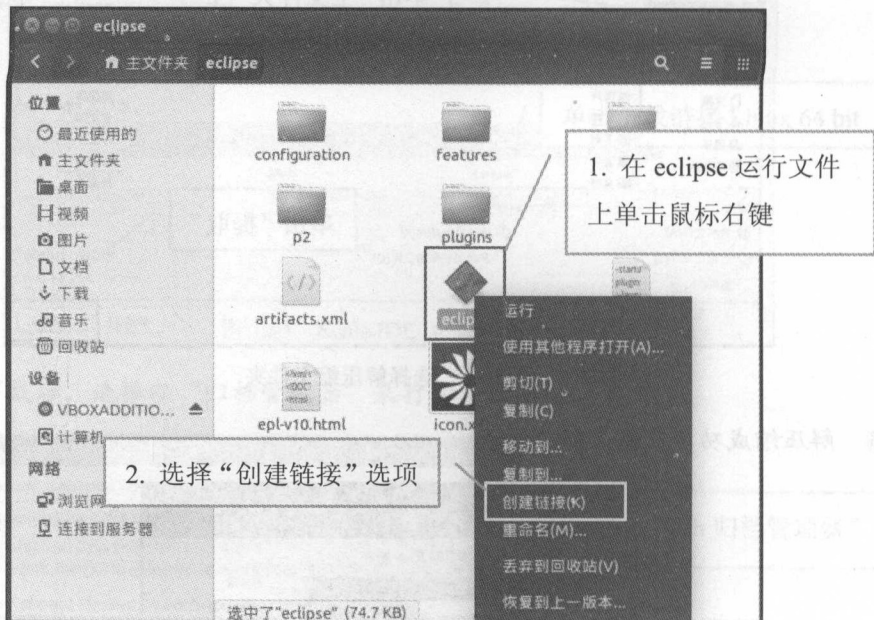


图 10-11 为 eclipse 创建链接

步骤 09 将创建的 eclipse 链接拖动到桌面

将创建的 eclipse 链接拖动到桌面。以后要运行 eclipse 时，只需要在图标上双击鼠标就行了，如图 10-12 所示。

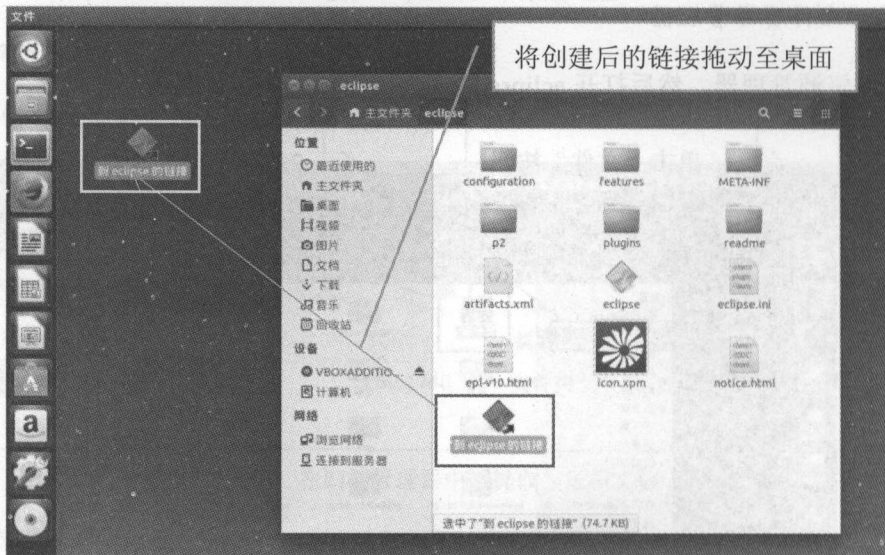


图 10-12 将创建的 eclipse 链接拖动到桌面

10.2 下载项目所需要的 Library

步骤 01 项目所需要的 Library

在后续章节中，我们开发 SPARK 应用程序需要以下链接库：

链接库	说明
spark-assembly	此链接库在我们安装 spark 时，就包含在 Spark 的 lib 目录下： /usr/local/spark/lib/spark-assembly-1.4.0-hadoop2.6.0.jar；此链接库主要负责 SPARK 与 Hadoop 沟通

另外，在程序中必须计算时间、绘图，所以也需要这些链接库。

链接库	说明
joda-Time	Java 链接库，主要用于处理包括 ISO8601 标准的 date 和 time
jfreeChart	Java 平台上的一个开放源码绘图链接库，它完全以 Java 语言编写。JFreeChart 可生成饼图、柱形图、散点图、时序图、甘特图等多种图表，并且可以产生 PNG 和 JPEG 格式的输出，还可以产生 PDF
jcommon	jfreeChart 需要 jcommon 才能运行

步骤 02 创建 Lib 目录

在“终端”程序中输入下列命令：

➤ 创建 Lib 目录用于存放链接库

workspace 是 eclipse 的工作区，我们再创建子目录 Lib：

```
mkdir -p ~/workspace/Lib
```

运行后屏幕显示界面如图 10-13 所示。

```
hduser@master: ~  
hduser@master:~$ mkdir -p ~/workspace/Lib
```

图 10-13 创建 Lib 目录

步骤 03 复制 spark Hadoop jar

我们可以在 Spark 安装目录复制此链接库，在“终端”程序中输入下列命令：

➤ 复制 spark-assembly-1.4.0-hadoop2.6.0.jar 至~/workspace/Lib 目录

```
sudo cp /usr/local/spark/lib/spark-assembly-1.4.0-hadoop2.6.0.jar  
~/workspace/Lib
```

运行后屏幕显示界面如图 10-14 所示。

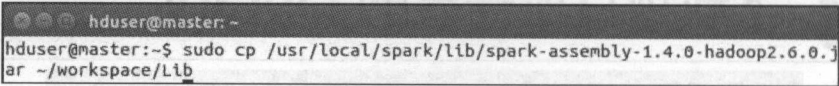


图 10-14 复制 spark Hadoop jar

步骤 04 搜索 jar 文件

可以在下列网站，找到 joda-time、jfreechart、jcommon jar。

➤ 在浏览器输入下载 jar 网站的网址：

www.Java2s.com

网站界面如图 10-15 所示，可以使用搜索功能，找到需要的链接库，例如搜索 joda-time。

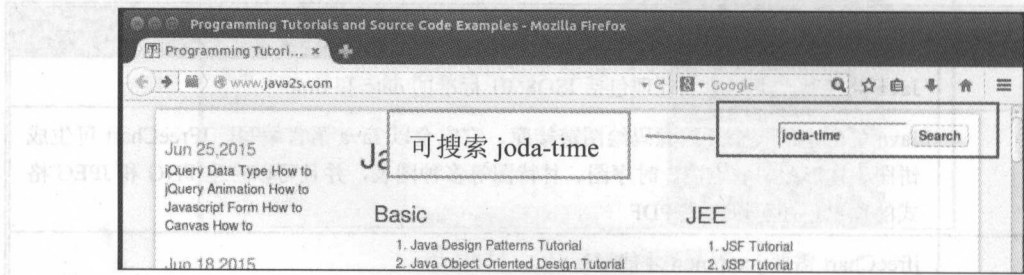


图 10-15 在网站搜索所需的 jar 文件

步骤 05 搜索 joda-time（见图 10-16）

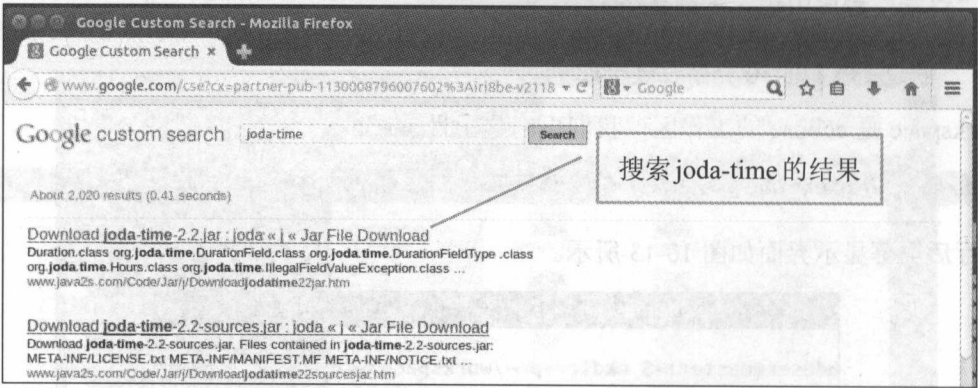


图 10-16 在网上搜索 joda-time 的结果

我们已经帮助准备好这些 jar 的下载链接，只需要按照下列步骤下载即可：

步骤 06 下载 joda-time

在“终端”程序中输入下列命令：

➤ 下载 joda-time，并且解压缩到~/workspace/Lib 目录


```
cd ~/workspace/Lib
wget http://www.Java2s.com/Code/JarDownload/joda/joda-time-2.2.jar.zip
unzip -j joda-time-2.2.jar.zip
```

执行后屏幕显示界面如图 10-17 所示。

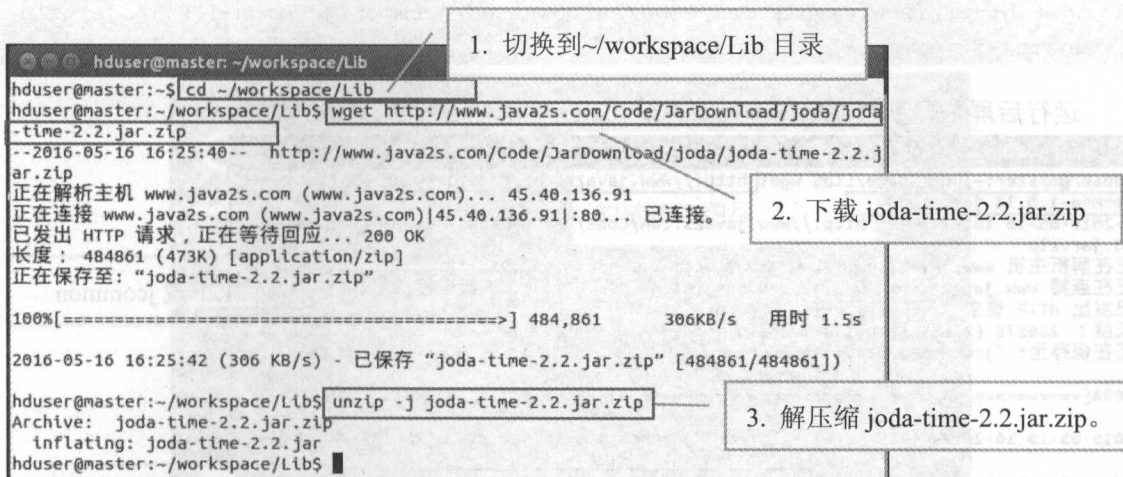


图 10-17 下载 joda-time

步骤 07 下载 jfreechart

在“终端”程序中输入下列命令：

➤ 下载并且解压缩 jfreechart，并且解压缩到~/workspace/Lib 目录中

```
wget http://www.Java2s.com/Code/JarDownload/jfreechart/jfreechart-1.0.3.jar.zip
unzip -j jfreechart-1.0.3.jar.zip
```

运行后屏幕显示界面如图 10-18 所示。

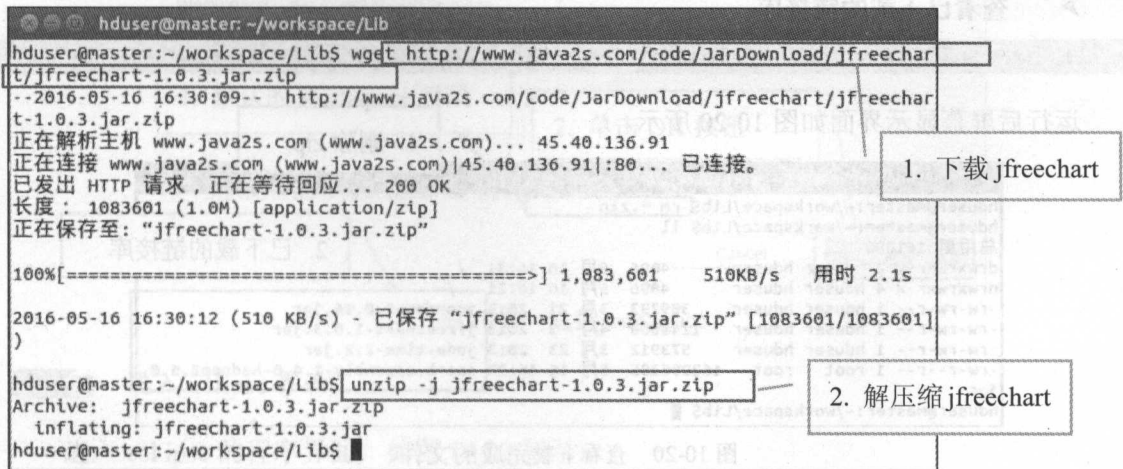


图 10-18 下载 jfreechart 并解压缩

步骤08 下载 jcommon

在“终端”程序中输入下列命令：

➤ 下载并且解压缩 jcommon 到~/workspace/Lib 目录

```
wget http://www.Java2s.com/Code/JarDownload/jcommon/jcommon-1.0.16.jar.zip
unzip -j jcommon-1.0.16.jar.zip
```

运行后屏幕显示界面如图 10-19 所示。

```
hduser@master: ~/workspace/Lib
hduser@master:~/workspace/Lib$ wget http://www.java2s.com/Code/JarDownload/jcommon/jcommon-1.0.16.jar.zip
--2016-05-16 16:27:22-- http://www.java2s.com/Code/JarDownload/jcommon/jcommon-1.0.16.jar.zip
正在解析主机 www.java2s.com (www.java2s.com)... 45.40.136.91
正在连接 www.java2s.com (www.java2s.com)|45.40.136.91|:80... 已连接。
已发出 HTTP 请求，正在等待回应... 200 OK
长度：286175 (279K) [application/zip]
正在保存至：“jcommon-1.0.16.jar.zip”

100%[=====>] 286,175      219KB/s  用时 1.3s

2016-05-16 16:27:24 (219 KB/s) - 已保存“jcommon-1.0.16.jar.zip” [286175/286175]

hduser@master:~/workspace/Lib$ unzip -j jcommon-1.0.16.jar.zip
Archive: jcommon-1.0.16.jar.zip
  inflating: jcommon-1.0.16.jar
hduser@master:~/workspace/Lib$
```

1. 下载 jcommon

2. 解压缩 jcommon

图 10-19 下载 jcommon 并解压缩

步骤09 下载完成的文件

在“终端”程序中输入下列命令：

➤ 删除 zip 以节省空间

```
rm *.zip
```

➤ 查看已下载的链接库

```
ll
```

运行后屏幕显示界面如图 10-20 所示。

```
hduser@master: ~/workspace/Lib
hduser@master:~/workspace/Lib$ rm *.zip
hduser@master:~/workspace/Lib$ ll
总用量 161080
drwxrwxr-x 2 hduser hduser      4096  5月 16 16:31 ./
drwxrwxr-x 4 hduser hduser      4096  5月 16 16:21 ../
-rw-rw-r-- 1 hduser hduser    309293  3月 21 2013 jcommon-1.0.16.jar
-rw-rw-r-- 1 hduser hduser  1148008  4月  3 2013 jfreechart-1.0.3.jar
-rw-rw-r-- 1 hduser hduser    573912  3月 23 2013 joda-time-2.2.jar
-rw-r--r-- 1 root  root    162896305  5月 16 16:22 spark-assembly-1.4.0-hadoop2.6.0.jar
hduser@master:~/workspace/Lib$
```

1. 删除 zip

2. 已下载的链接库

图 10-20 查看下载完成的文件

10.3 启动 eclipse

接下来我们将启动 eclipse。

步骤 01 用鼠标双击链接至 eclipse 的图标（见图 10-21）

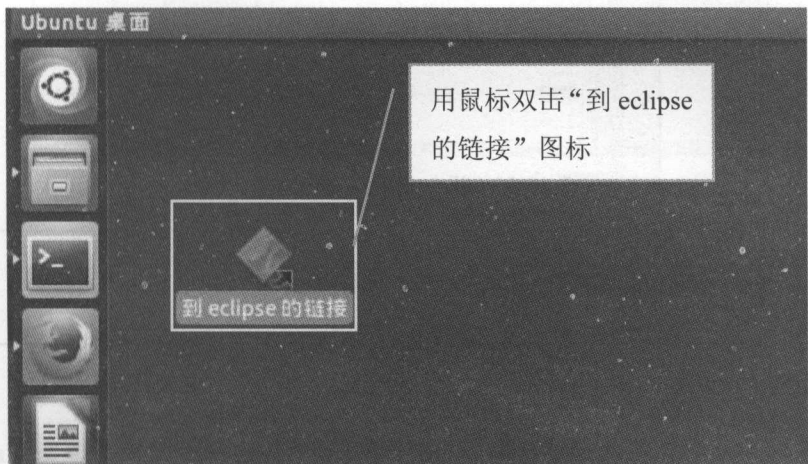


图 10-21 用鼠标双击“到 eclipse 的链接”图标

步骤 02 选择 workspace

运行 eclipse 后会出现对话框，让你选择工作目录。默认是/home/hduser/workspace 目录，也可以选择其他的工作目录，如图 10-22 所示。

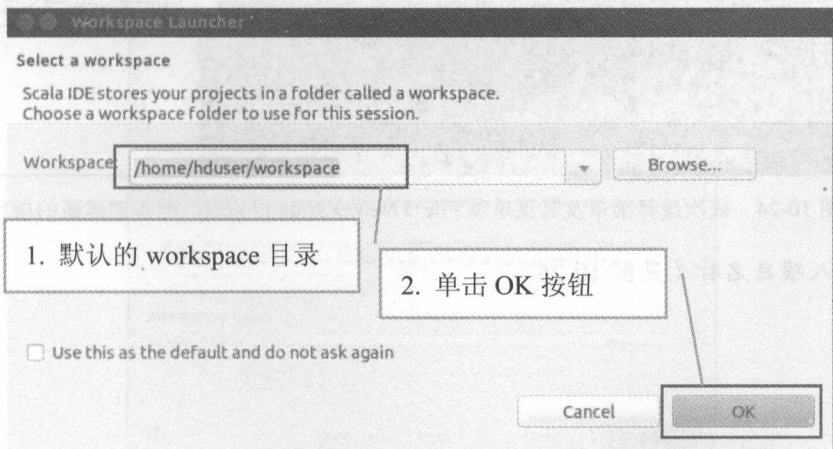


图 10-22 选择 workspace

步骤 03 进入 eclipse 界面

进入 eclipse 的程序界面，如图 10-23 所示。

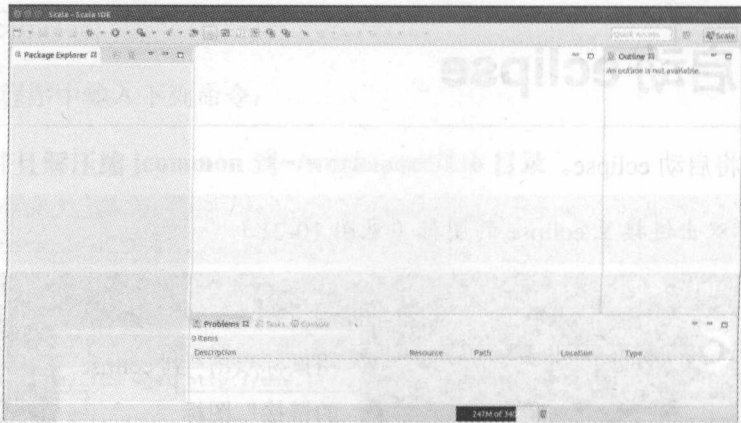


图 10-23 进入 eclipse 的程序界面

10.4 创建新的 Spark 项目

本节将创建 WordCount 项目。

步骤 01 依次选择菜单及其选项：File→New→Scala Project（见图 10-24）

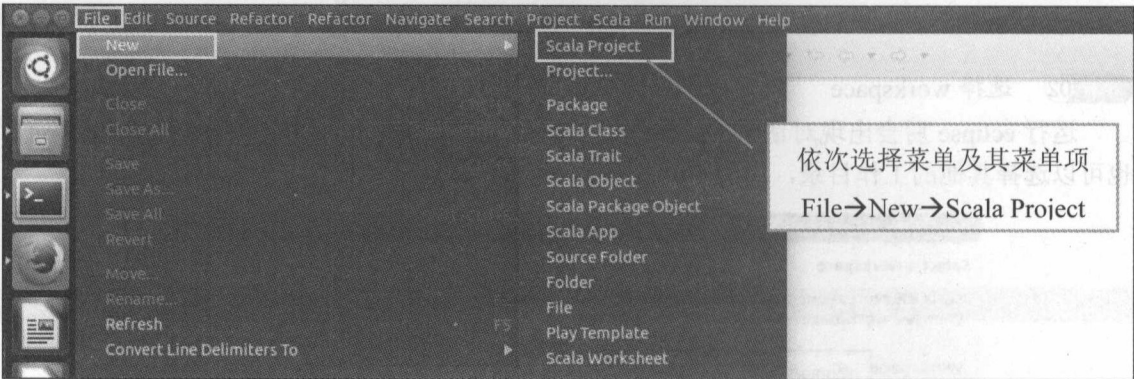


图 10-24 依次选择菜单及其菜单项 File→New→Scala Project，准备创建新的项目

步骤 02 输入项目名称（见图 10-25）

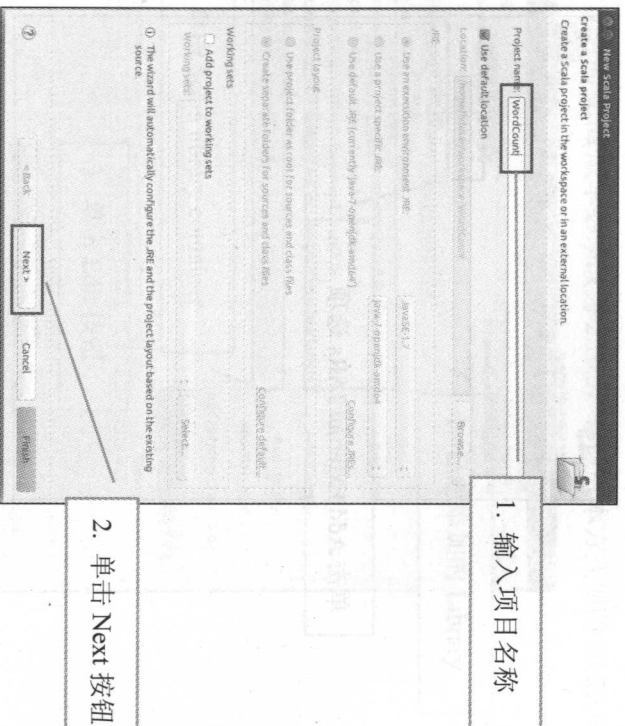


图 10-25 输入项目名称

步骤 03 Scala 设置界面 (见图 10-26)

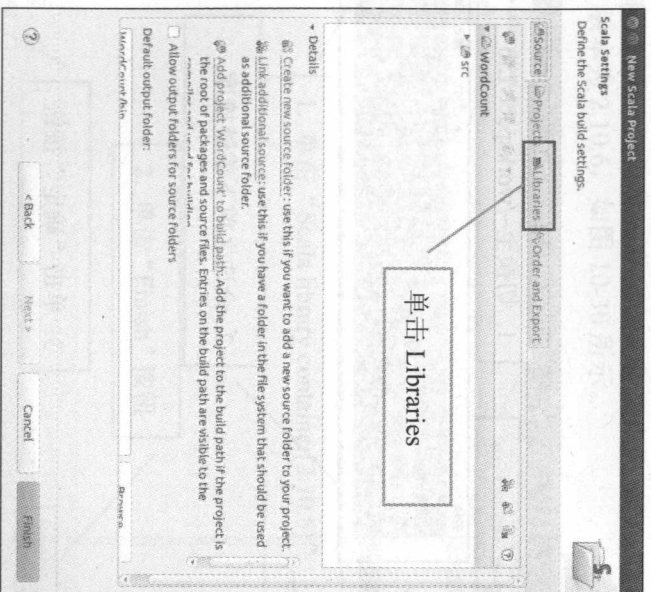


图 10-26 Scala 的设置界面

步骤 04 添加外部 JAR (见图 10-27)

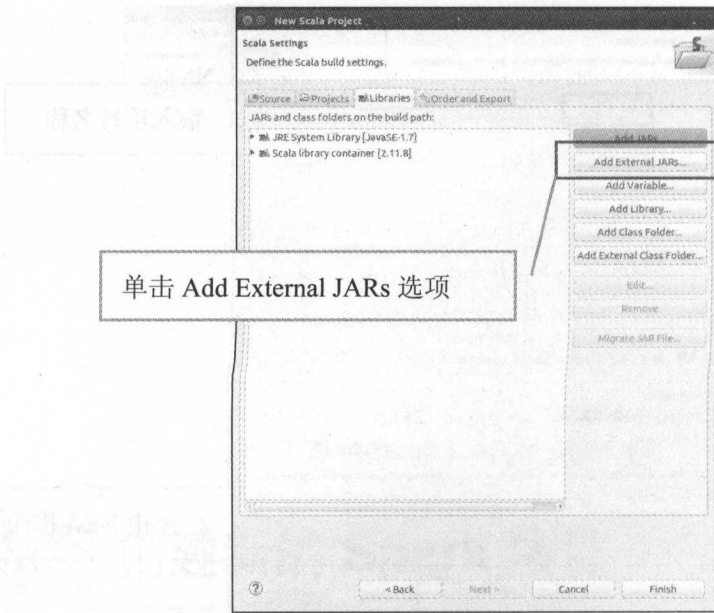


图 10-27 添加外部 JAR

步骤 05 选择之前下载的 JAR 链接库

单击 Add External JARs 后，会出现 JAR Selection 对话框，添加之前下载的链接库。如图 10-28 所示。

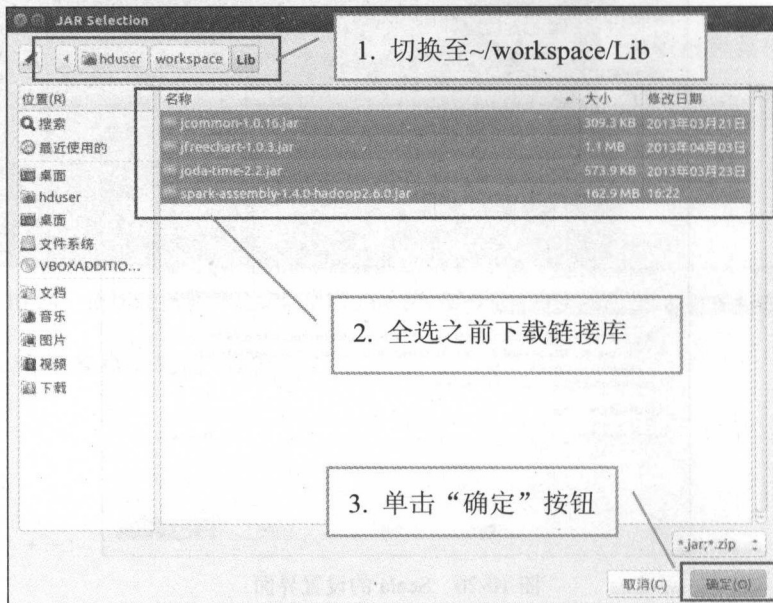


图 10-28 选择之前下载的 JAR 链接库

步骤 06 设置选用的 scala library 版本

在 Eclipse Scala IDE 中默认的 Scala library 版本是[2.11.6]，但是 Spark 在 Scala [2.11.6]版本会

出现编译错误，所以 Spark 配合的 scala 版本必须改为 2.10.5，修改版本方式如图 10-29 所示。

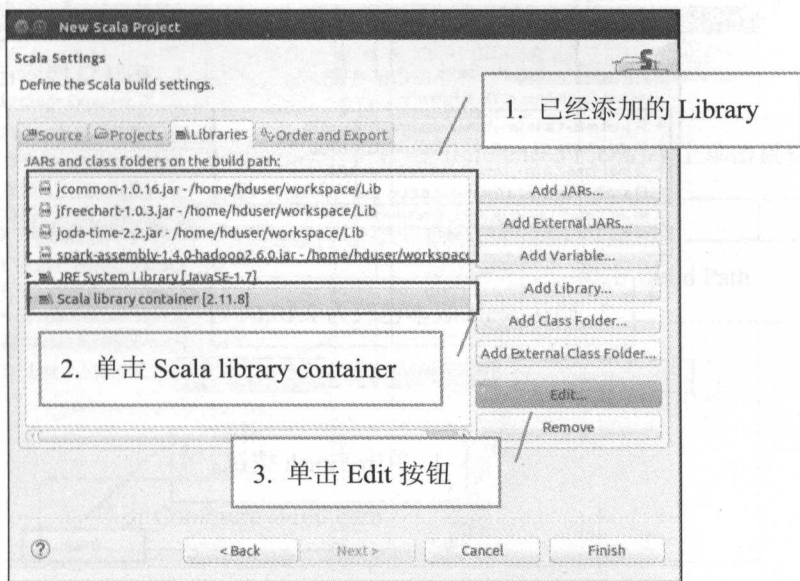


图 10-29 设置选用的 scala library 的版本

步骤 07 设置 scala library 版本为 2.10.6

设置 scala library 版本为 2.10.6，如图 10-30 所示。

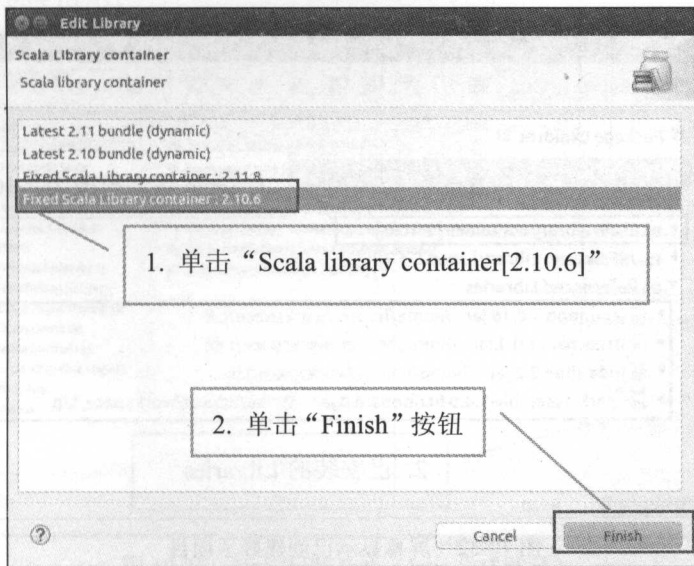


图 10-30 设置 scala library 版本为 2.10.6

步骤 08 完成设置

完成设置后回到 New Scala Project 界面。确认链接库已设置好了，并且 scala 版本已改为 2.10.6，如图 10-31 所示。

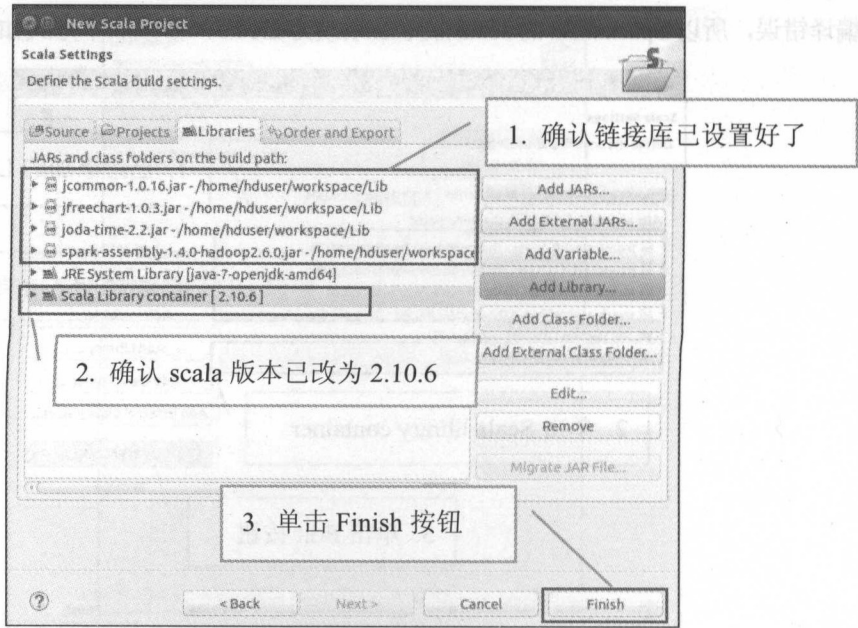


图 10-31 确定完成好了设置

步骤 09 已创建项目的界面

设置完成后，可以在 Package Explorer 看到 Scala 版本已修改，并已经安装了 Libraries，如图 10-32 所示。

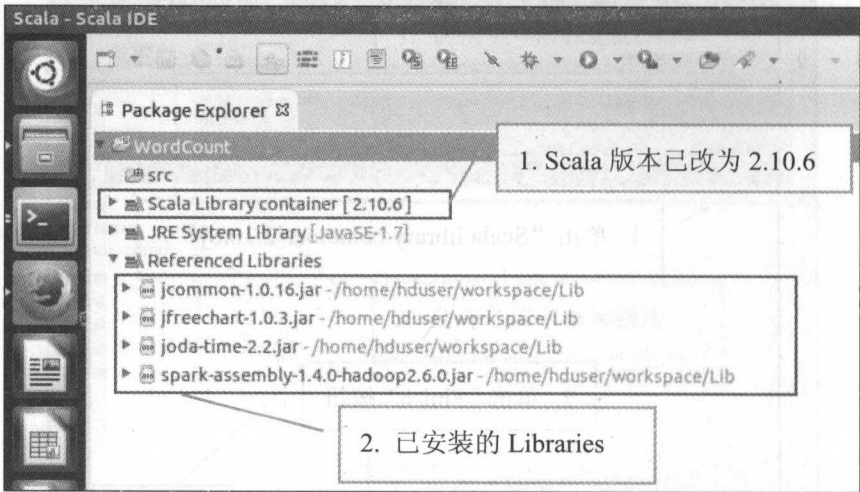


图 10-32 屏幕显示已创建好了项目

10.5 设置项目链接库

链接库除了可以在创建项目时设置，在项目创建后仍可以设置链接库。

步骤 01 Configure Build Path (见图 10-33)

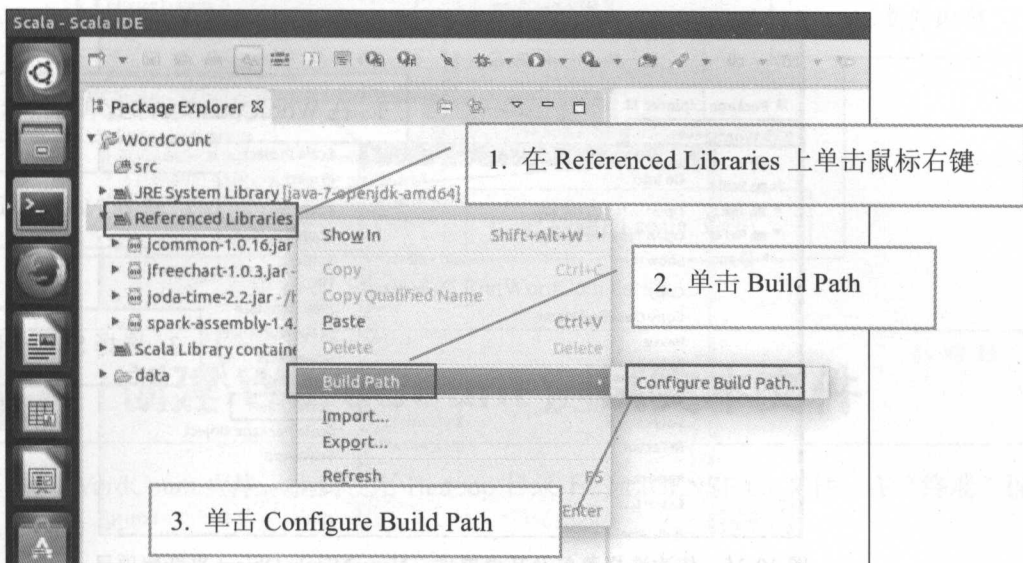


图 10-33 Configure Build Path (配置项目生成路径)

步骤 02 Java Build Path

单击 Configure Build Path 后, 就会出现 Java Build Path 设置界面, 设置方法与上一节设置方法相同, 如图 10-34 所示。

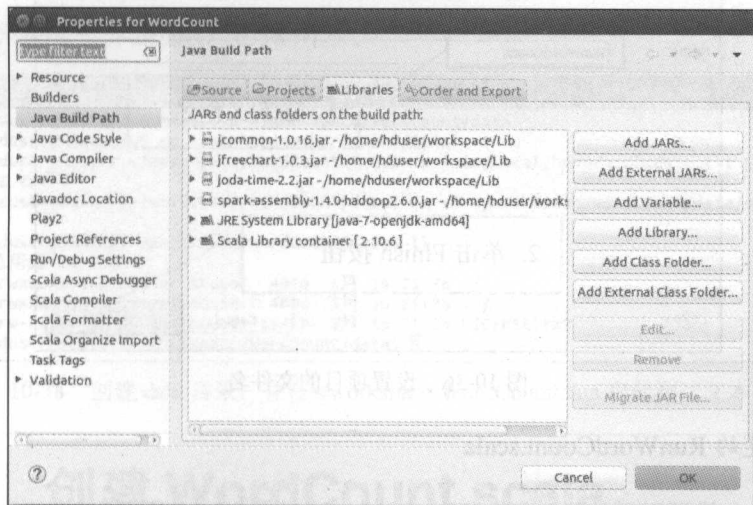


图 10-34 Java Build Path (Java 生成路径)

10.6 新建 scala 程序

接下来将创建 WordCount.scala 程序。

步骤 01 依次选择菜单及其菜单项：New→Scala Object（见图 10-35）

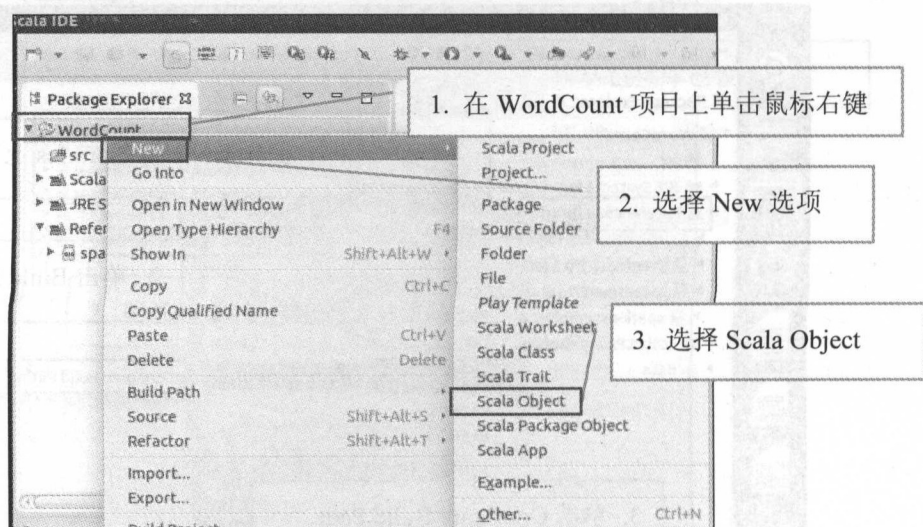


图 10-35 依次选择菜单及其菜单项：New→Scala Object 来新建项目

步骤 02 设置文件名（见图 10-36）

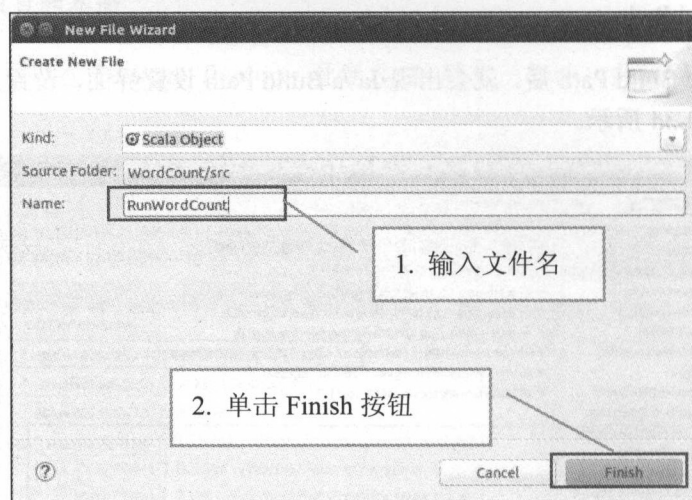


图 10-36 设置项目的文件名

步骤 03 已创建的 RunWordCount.scala

创建完成后，就可以看到 RunWordCount.scala，如图 10-37 所示。

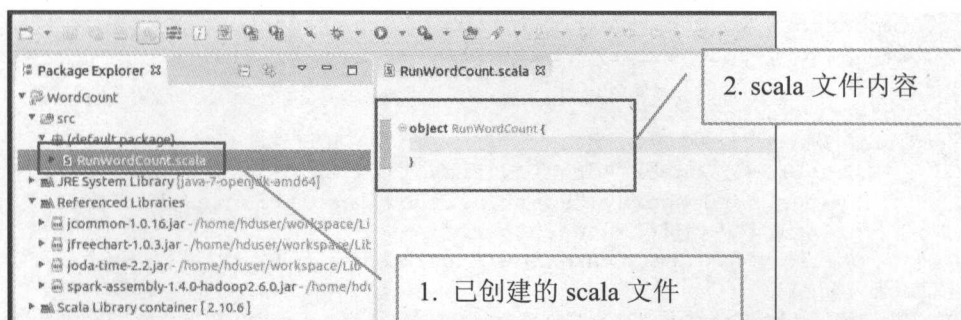


图 10-37 已创建的 RunWordCount.scala

10.7 创建 WordCount 测试文本文件

为了测试 WordCount 程序，我们使用 Hadoop 目录下的 LICENSE.txt 文件。在“终端”程序中输入下列命令：

➤ 创建 data 目录，并在~/workspace/WordCount/data 存储测试文本文件

```
mkdir -p ~/workspace/WordCount/data
cd ~/workspace/WordCount/data
cp /usr/local/hadoop/LICENSE.txt LICENSE.txt
rm -R ~/workspace/WordCount/data/output
ll
```

运行后，屏幕显示界面如图 10-38 所示。

```
hduser@master: ~/workspace/WordCount/data
hduser@master:~$ mkdir -p ~/workspace/WordCount/data
hduser@master:~$ cd ~/workspace/WordCount/data
hduser@master:~/workspace/WordCount/data$ cp /usr/local/hadoop/LICENSE.txt LICEN
SE.txt
hduser@master:~/workspace/WordCount/data$ rm -R ~/workspace/WordCount/data/outpu
t
hduser@master:~/workspace/WordCount/data$ ll
总用量 24
drwxrwxr-x 2 hduser hduser 4096 5月 16 21:36 ./
drwxrwxr-x 5 hduser hduser 4096 5月 16 21:35 ../
-rw-r--r-- 1 hduser hduser 15429 5月 16 21:36 LICENSE.txt
hduser@master:~/workspace/WordCount/data$
```

图 10-38 创建 data 目录，并在~/workspace/WordCount/data 存储测试文本文件

10.8 创建 WordCount.scala

步骤 01 输入 WordCount.scala 程序代码

在 eclipse 中输入 WordCount.scala 程序代码，如下所示：

```
import org.apache.log4j.Logger
import org.apache.log4j.Level
```

```
import org.apache.spark.{ SparkConf, SparkContext }
import org.apache.spark.rdd.RDD

object RunWordCount {
  def main(args: Array[String]): Unit = {
    Logger.getLogger("org").setLevel(Level.OFF)
    System.setProperty("spark.ui.showConsoleProgress", "false")
    println("开始运行RunWordCount")
    val sc = new SparkContext(new SparkConf().setAppName("wordCount").setMaster(
("local[4]"))
    println("开始读取文本文件...")
    val textFile = sc.textFile("data/LICENSE.txt")
    println("开始创建 RDD...")
    val countsRDD = textFile.flatMap(line => line.split(" "))
      .map(word => (word, 1))
      .reduceByKey(_ + _)
    println("开始保存到文本文件...")
    try {
      countsRDD.saveAsTextFile("data/output")
      println("已经存盘成功")
    } catch {
      case e: Exception => println("输出目录已经存在, 请先删除原有目录");
    }
  }
}
```

上述程序代码的详细情况请参考第 9 章，现在简略说明如下：

➤ 设置不要显示过多信息

```
import org.apache.log4j.Logger
import org.apache.log4j.Level
Logger.getLogger("org").setLevel(Level.OFF)
System.setProperty("spark.ui.showConsoleProgress", "false")
```

因为 spark 程序默认会显示很多信息，这些信息对于调试有帮助。不过信息太多也会让程序运行界面很乱，我们可以设置不要显示这些信息。这些信息是以 log4j 开发的，所以需要与 import log4j 相关的 lib。

➤ 导入相关链接库

```
import org.apache.spark.{ SparkConf, SparkContext }
import org.apache.spark.rdd.RDD
```

导入 SparkContext 与 RDD 链接库。

➤ 创建 SparkContext

```
val sc = new SparkContext(new SparkConf().
setAppName("wordCount").setMaster("local[4]"))
```

接下来创建 SparkContext。其中 setMaster("local[4]") 设置为 local 运行（即本地运行）。虽然是本地运行，因为目前大部分计算机的 CPU 都是多核的，所以仍可多线程运行以加快运

行速度。后续章节会介绍如何在 Hadoop multi-cluster 运行程序。

➤ 读取文本文件

```
val textFile = sc.textFile("data/LICENSE.txt")
```

使用 `sc.textFile` 读取文本文件

➤ 执行 Map/Reduce 运算

```
val countsRDD = textFile
  .flatMap(line => line.split(" "))
  .map(word => (word, 1))
  .reduceByKey(_ + _)
```

创建 `countsRDD`:

1. 以空格符分隔单词，读取每一个英文单词。
2. 以 `map` 转换为 Key-Value (`word,1`)。
3. 最后使用 `reduceByKey` 将相同的 `key` 值相加。

➤ 存储文件

```
try {
  countsRDD.saveAsTextFile("data/output")
  println("已经存盘成功")
} catch {
  case e: Exception =>
    println("输出目录已经存在，请先删除原有目录");
}
```

`countsRDD` 使用 `saveAsTextFile` 将结果存储至目录，在此使用 `try catch` 语句。如果输出目录已经存在就会发生错误，并显示错误信息。

10.9 编译 WordCount.scala 程序

前面章节中已经完成了程序的编辑，现在需要先编译程序才能够运行。

步骤 01 修改 Eclipse 的默认编译模式

Eclipse 默认的编译模式是 Build Automation，所以可以看到菜单上 Build Automation 前面有打勾的符号。也就是说当将程序存盘时会自动编译。这样的好处是可以实时发现编译的错误。但是，编译需要耗费 CPU 的计算资源，可能会影响系统当前的工作。也可以选择不要自动编译，等程序全部修改完成之后再自行编译，如图 10-39 所示。

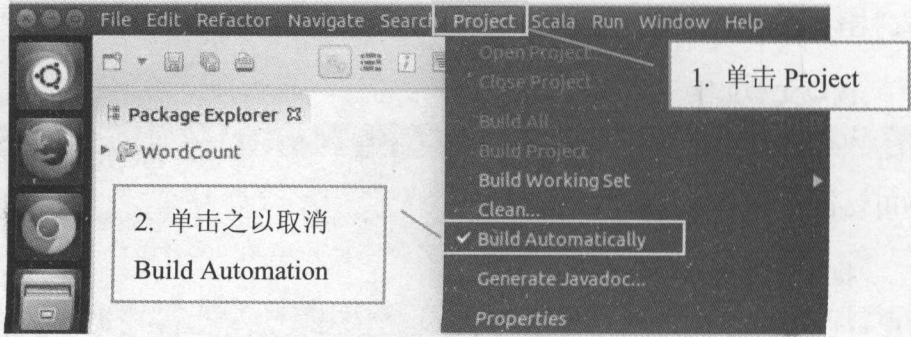


图 10-39 修改 Eclipse 的默认编译模式

步骤 02 编译项目（见图 10-40）

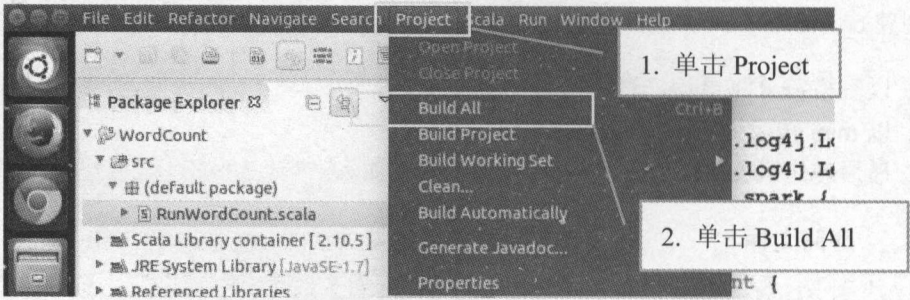


图 10-40 编译项目

步骤 03 编译错误

如果编译错误，会显示在 Problems 窗口中，如图 10-41 所示。

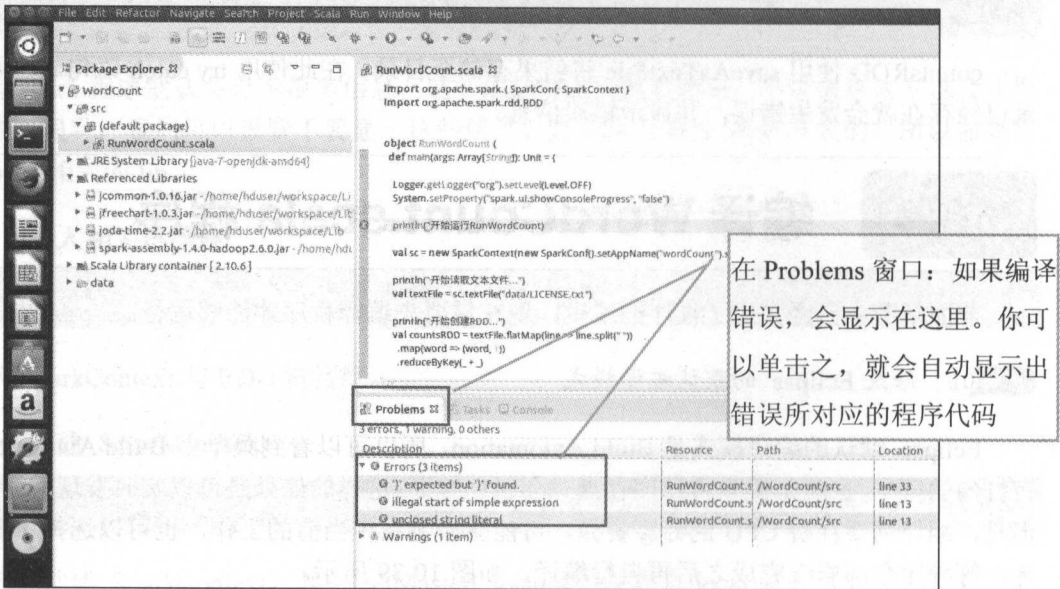


图 10-41 编译错误显示在 Problems 窗口中的情况

步骤 04 编译完成

当把所有的错误修改完成再次执行编译时，如果没有再出现任何错误（Errors），基本上程序的编译就已经完成了。界面上只会剩下 Warning 警告信息，但是它们并不影响程序的编译与运行，可以继续修改或是忽略它，如图 10-42 所示。

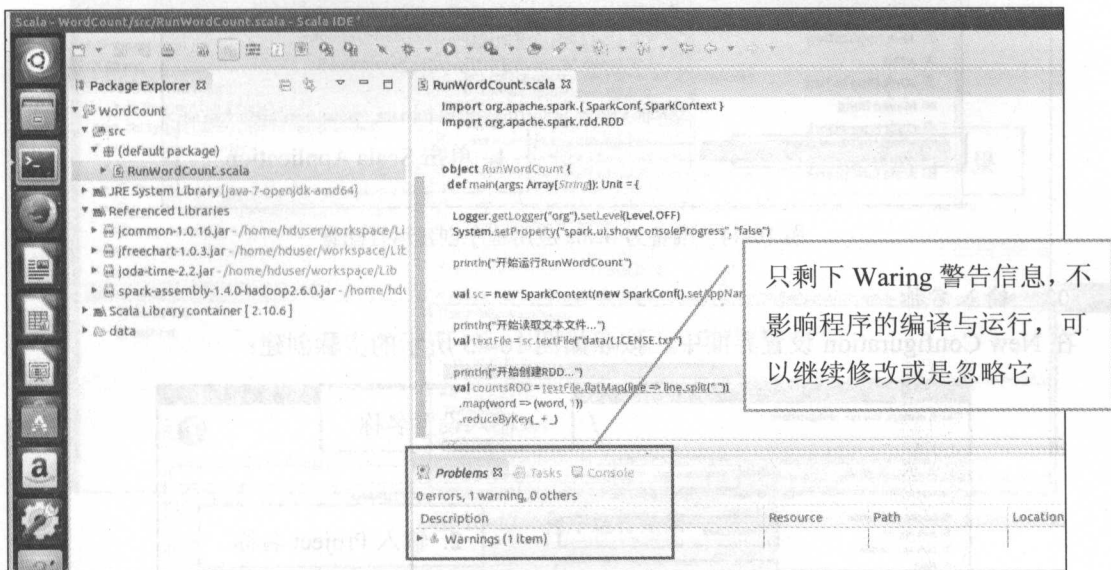


图 10-42 编译完成，只剩下警告信息

10.10 运行 WordCount.scala 程序

步骤 01 依次选择菜单及其菜单项：Run→Run Configurations

运行程序前，必须先进行 Run Configurations 设置，如图 10-43 所示。

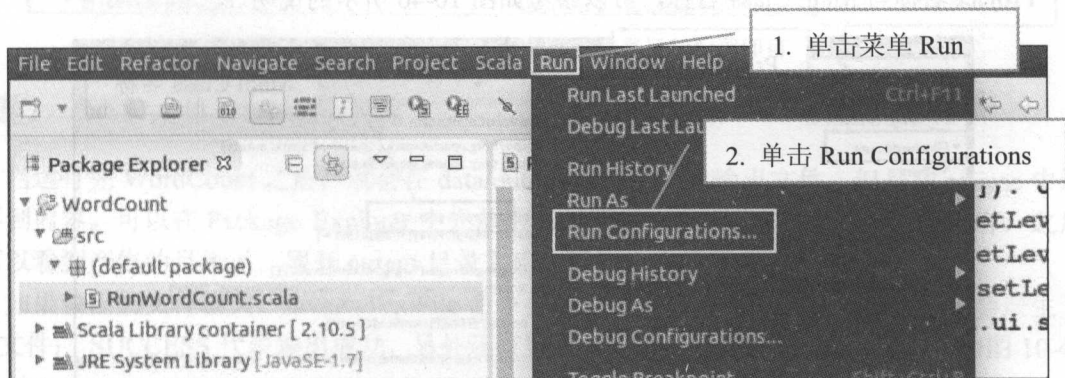


图 10-43 准备进行 Run Configurations（运行配置）的设置

步骤 02 单击新建图标（见图 10-44）

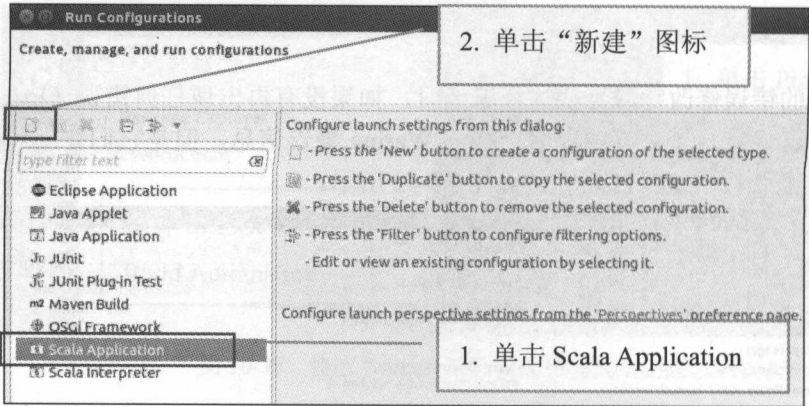


图 10-44 准备为 Scala 应用程序创建运行配置

步骤 03 输入名称

在 New Configuration 设置界面中，按照如图 10-45 所示的步骤创建：

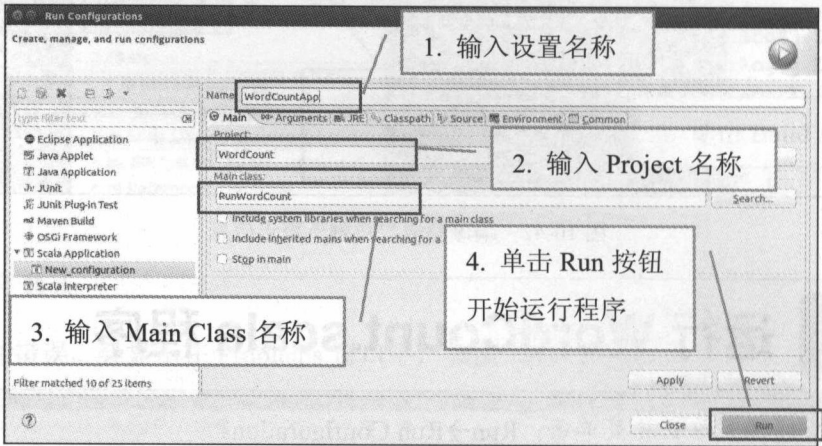


图 10-45 创建运行配置

Project 名称与 Main Class 名称，可以参考如图 10-46 所示的说明。

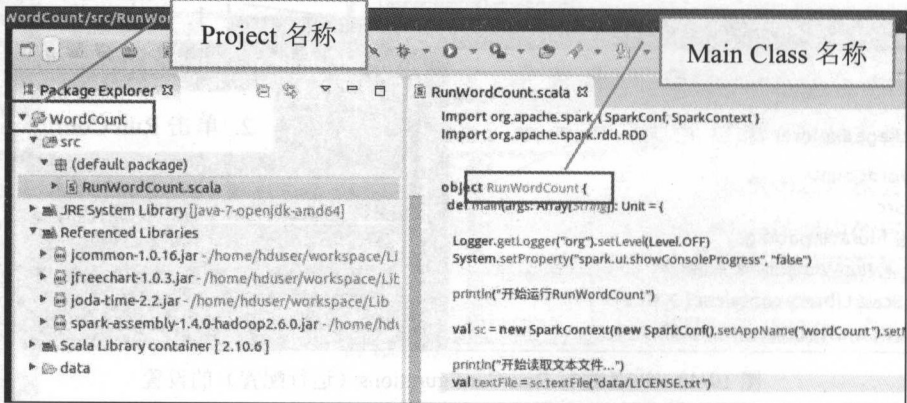


图 10-46 Project 名称与 Main Class 名称

步骤 04 运行结果

单击 Run 按钮之后，程序就会开始运行，如图 10-47 所示。

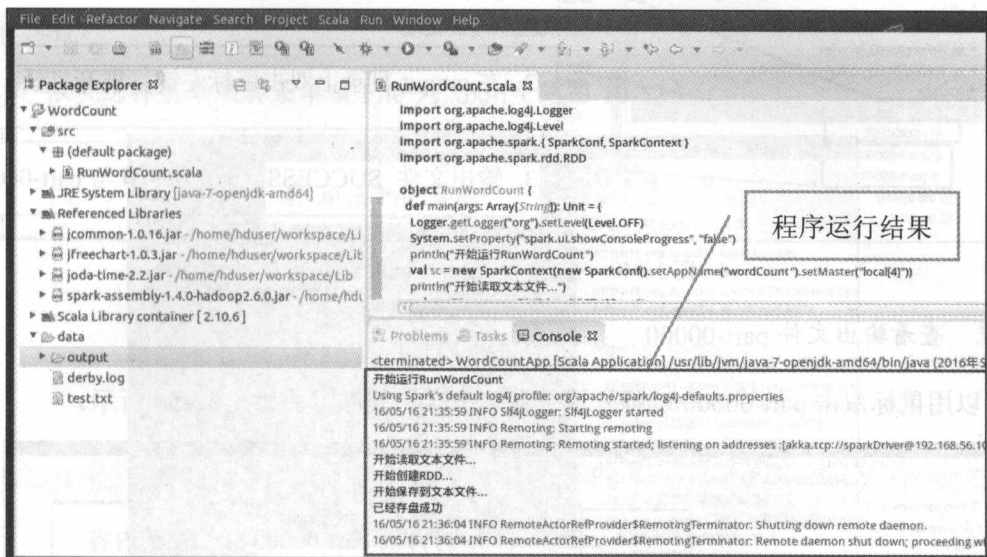


图 10-47 程序运行结果

步骤 05 再次运行程序

因为先前已经创建了运行配置（Run Configuration），下次如果要再次运行程序时，就不需要再重复设置了。按照下列步骤运行，如图 10-48 所示。

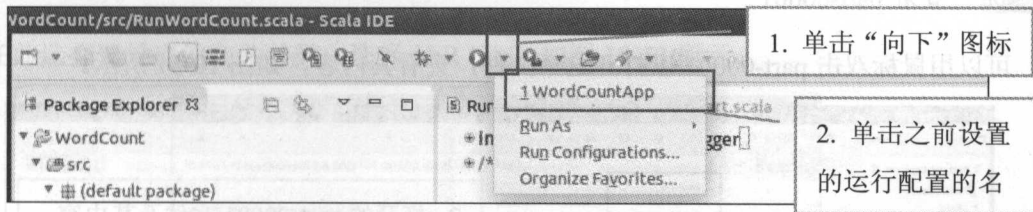


图 10-48 再次运行程序之前只需选择运行配置即可

步骤 06 查看输出的 data/output 目录

当运行完 WordCount 之后，就会在 data/output 目录中产生输出文件。但是在 eclipse 中还看不到内容。可以在 Package Explorer 中先单击 data 目录，然后按 F5 键更新目录内容，之后就可以看到产生的目录了。展开 output 目录之后会看到输出的文件。

如果输出的文件很大，saveAsTextFile 命令会自动把它分为多个文件。本范例就会产生了 3 个文件：_SUCCESS 代表输出成功，另外还有数据文件：part-00000 与 part-00001，如图 10-49 所示。

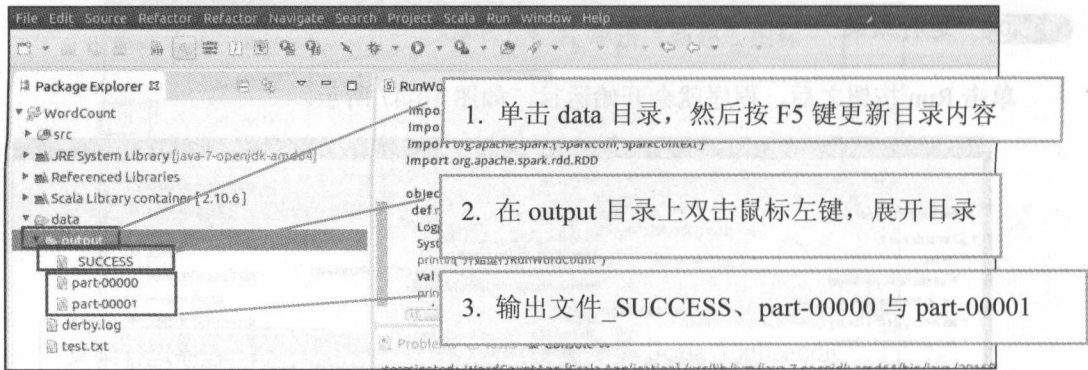


图 10-49 查看输出的 data/output 目录

步骤 07 查看输出文件 part-00000

可以用鼠标双击 part-00000 以打开这个文件，查看其内容，如图 10-50 所示。

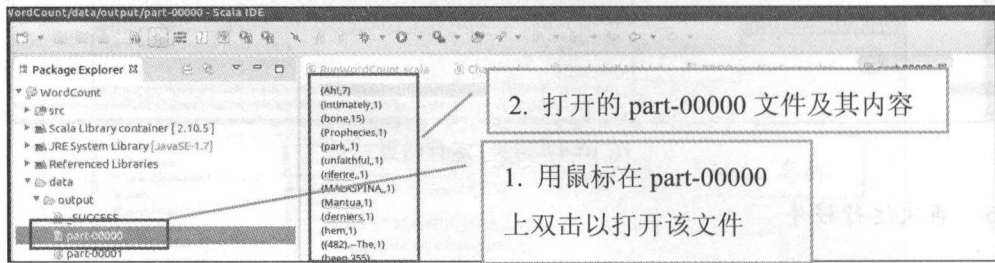


图 10-50 查看输出文件 part-00000

步骤 08 查看 part-00001

可以用鼠标双击 part-00001 以打开这个文件，查看其内容，如图 10-51 所示。

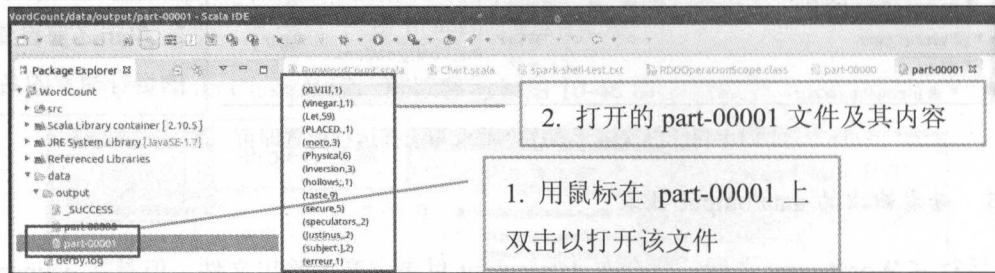


图 10-51 查看输出文件 part-00001

10.11 导出 jar 文件

我们在 eclipse 集成开发环境中开发程序很方便，如果希望将开发的程序交给其他的用户运行，而其他的用户可能没有安装 eclipse，怎么办？当我们要交付程序给用户运行时，并不是给很多零散的.class 文件，而是将编译好的.class 文件打包成扩展名为.jar 的文件，然后用户

就可以启动 JAR (Java Archive File) 文件来运行程序。

打包成 JAR 文件之后,在 spark 的 bin 目录下(/usr/local/spark/bin),附带有 spark-submit 工具程序。可以使用 spark-submit,在不同的环境下运行 Spark 的 jar 程序。按照下列步骤即可导出 jar 文件:

步骤 01 依次选择菜单及其菜单项 File→Export (见图 10-52)

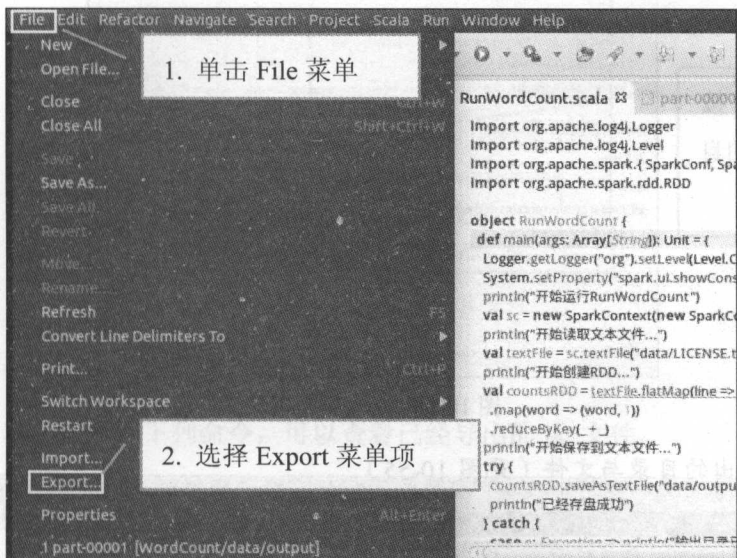


图 10-52 依次选择菜单及其菜单项 File→Export, 准备导出

步骤 02 选择导出 JAR File

在此选择要导出的文件种类,选择 JAR File,如图 10-53 所示。

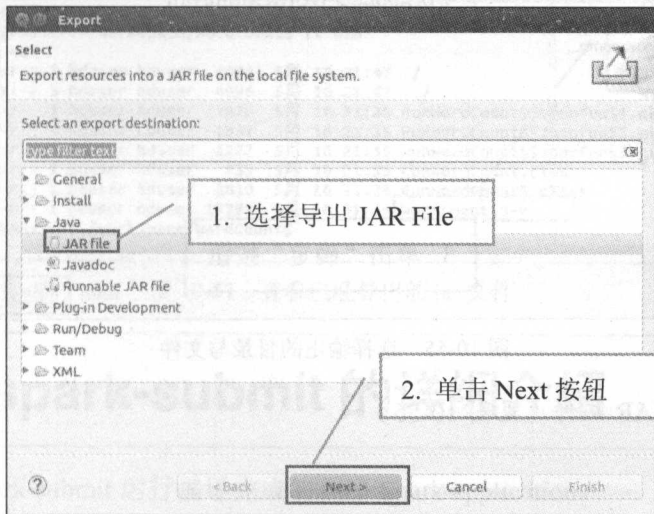


图 10-53 选择导出 JAR File

步骤 03 JAR 文件设置 (见图 10-54)

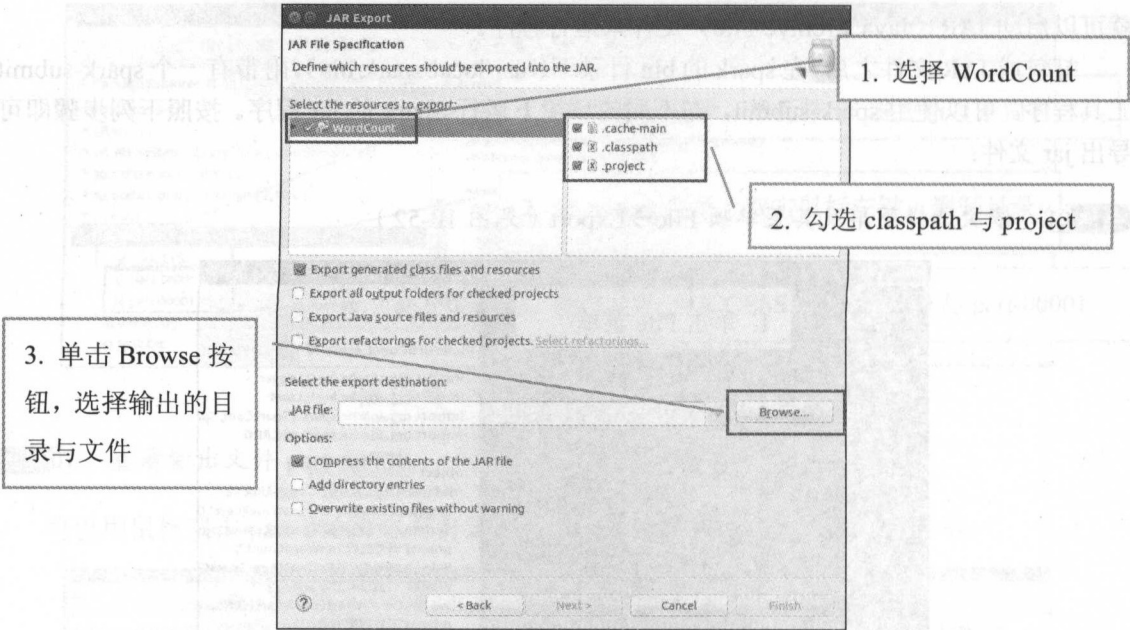


图 10-54 JAR 文件设置

步骤 04 选择输出的目录与文件（见图 10-55）

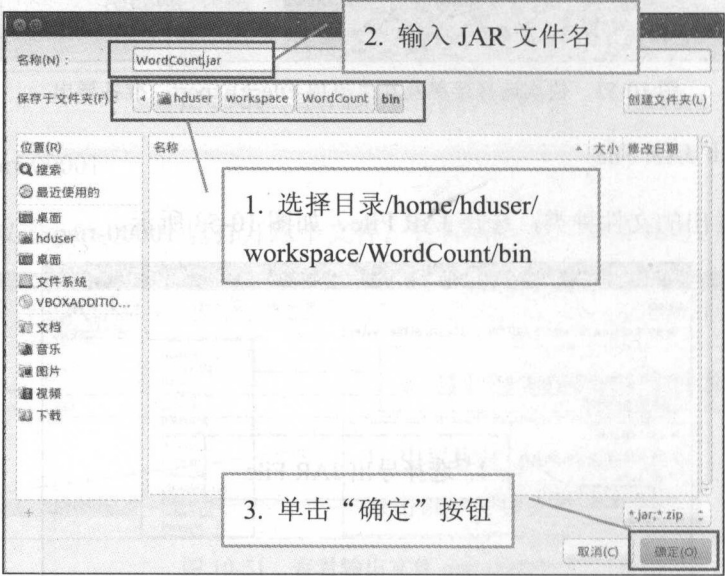


图 10-55 选择输出的目录与文件

步骤 05 开始导出 JAR 文件（见图 10-56）

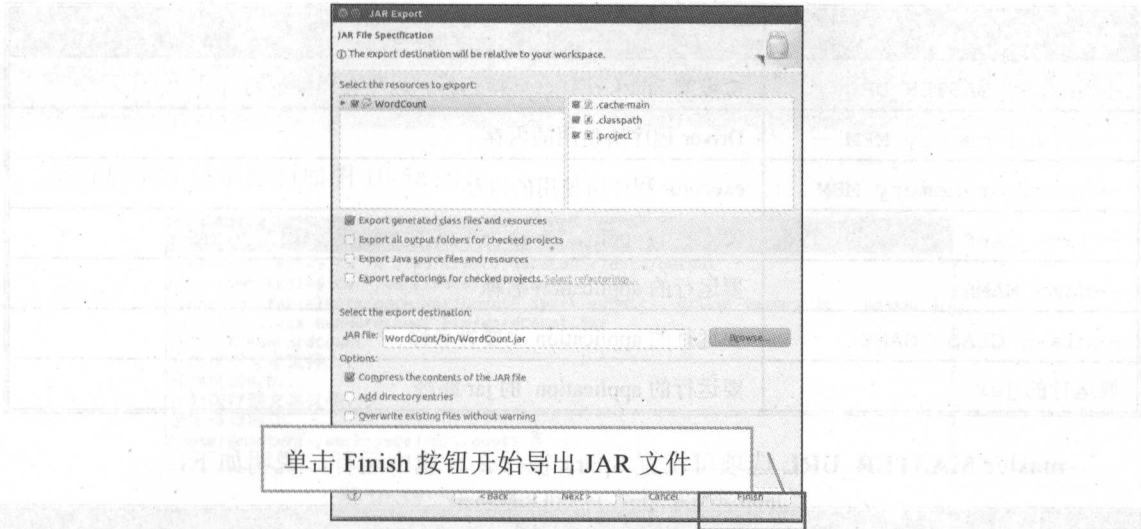


图 10-56 开始导出 JAR 文件

步骤 06 查看已经导出的 jar 文件

在“终端”程序中输入下列命令，可以查看已经导出的 jar 文件。

命令	说明
cd ~/workspace/WordCount	切换至 WordCount 项目目录
ll bin	查看 bin 目录

运行后屏幕显示界面如图 10-57 所示。

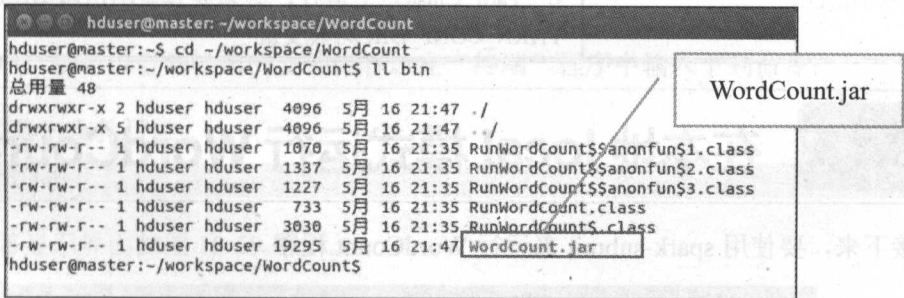


图 10-57 查看已经导出的 jar 文件

10.12

spark-submit 的详细介绍

接下来，以 spark-submit 运行编译完成的程序 Spark application。
spark-submit 常用的选项如下：

选项	说明
--master MASTER_URL	可设置 Spark 在什么环境运行，以下会详细说明
--driver-memory MEM	Driver 程序所使用的内存
--executor-memory MEM	executor 程序所使用的内存
--jars JARS	要运行的 application 会引用到的外部链接库
--name NAME	要运行的 application 名称
--class CLASS_NAME	要运行的 application 主要类名称
要运行的 jar	要运行的 application 的 jar 路径

--master MASTER_URL 选项可设置 Spark 在什么环境中运行，说明如下：

MASTER URL	说明
Local	在本地运行；只使用一个线程
local[K]	在本地运行；使用 K 个线程（会使用本地计算机的多核 CPU）
local[*]	在本地运行；Spark 会自动尽量利用本地计算机上的多核 CPU
spark://HOST:PORT	在 Spark Standalone Cluster 上运行；例如 spark://master:7077（默认 port 是 7077）
mesos://HOST:PORT	在 Mesos cluster 运行上（默认 port 是 5050）
yarn-client	在 Yarn Client 上运行，必须要设置 HADOOP_CONF_DIR or YARN_CONF_DIR 环境变量
yarn-cluster	在 Yarn Cluster 上运行，必须要设置 HADOOP_CONF_DIR or YARN_CONF_DIR 环境变量

10.13 在本地 local 模式运行 WordCount 程序

接下来，要使用 spark-submit 来运行 WordCount 程序。

步骤 01 在本地（local）运行 WordCount

在“终端”程序中输入下列命令，在本地（local）运行 WordCount：

➤ 删除之前的 data/output 目录

```
rm -R ~/workspace/WordCount/data/output
```

➤ 切换至 WordCount 项目目录

```
cd ~/workspace/WordCount
```

在本地运行 WordCount

```
spark-submit --driver-memory 2g --master local[4] --class RunWordCount
bin/WordCount.jar
```

运行后屏幕显示界面如图 10-58 所示。

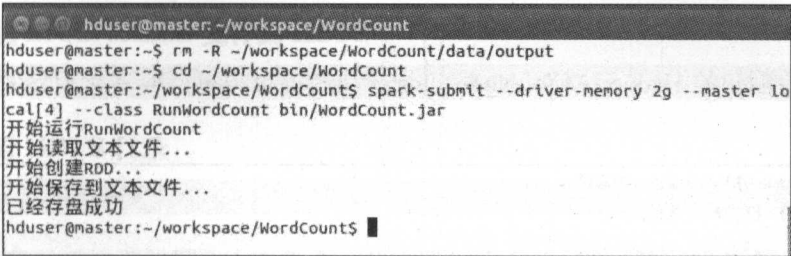


图 10-58 在本地（local）运行 WordCount

以上 spark-submit 命令详细说明如下：

命令	说明
spark-submit	spark-submit 命令
--driver-memory 2g	设置 driver 程序使用 2G 的内存
--master local[4]	在本地运行；使用 4 个线程（会使用本地计算机上的多核 CPU）
--class RunWordCount	设置 main 类
bin/WordCount.jar	WordCount Jar 路径

步骤 02 查看输出文件目录

运行完成后，就可以看到输出的文件内容。在“终端”程序中输入下列命令：

查看输出文件目录

```
ll data/output
```

运行后屏幕显示界面如图 10-59 所示。

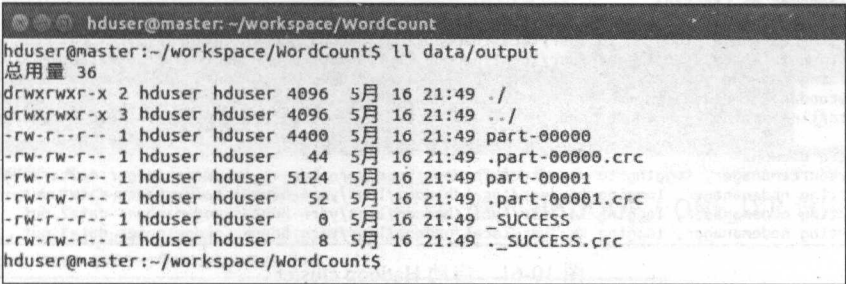


图 10-59 查看输出文件目录

10.14 在 Hadoop yarn-client 运行 WordCount 程序

接下来要介绍如何使用 spark-submit，在 Hadoop YARN 上运行 WordCount Spark 程序。

步骤 01 启动 VirtualBox 虚拟机

先启动之前创建的 Hadoop Multi Node Cluster 所有的虚拟机，屏幕显示界面如图 10-60 所示。

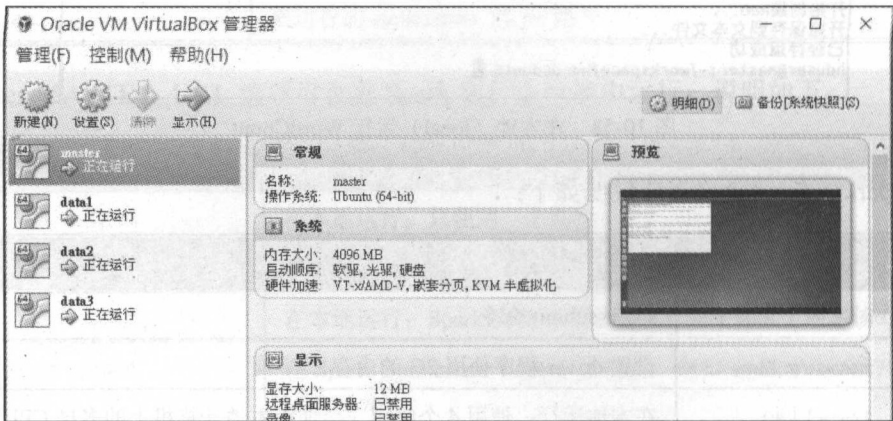


图 10-60 启动 VirtualBox 虚拟机

步骤 02 启动 Hadoop cluster

在 master 服务器的“终端”程序中输入下列命令：

➤ 启动 Hadoop 集群 cluster

```
start-all.sh
```

运行后屏幕显示界面如图 10-61 所示。

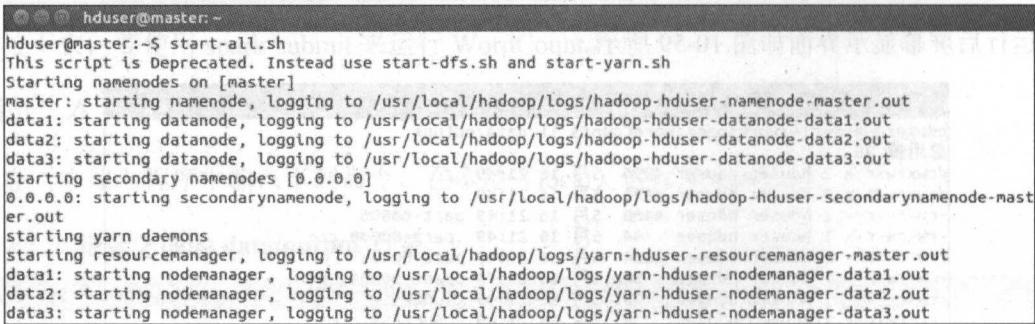


图 10-61 启动 Hadoop cluster

步骤 03 复制测试文件到 HDFS

当 Spark 程序在 Hadoop YARN 上运行时，默认会读取 HDFS 文件。所以必须先将输入文件复制到 HDFS。在“终端”程序中输入下列命令：

➤ 在 HDFS 创建 data 目录

```
hadoop fs -mkdir data
```

➤ 复制 LICENSE.txt 文件到 HDFS

```
hadoop fs -copyFromLocal ~/workspace/WordCount/data/LICENSE.txt data
```

➤ 查看 HDFS 目录

```
hadoop fs -ls data
```

运行后屏幕显示界面如图 10-62 所示，从中可以看到已经复制到 HDFS 的测试文件。

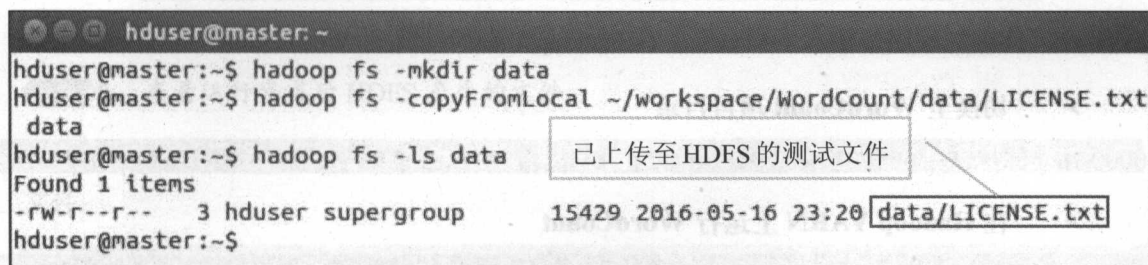


图 10-62 复制测试文件至 HDFS

步骤 04 修改环境配置文件

在 Hadoop YARN 上运行 Spark 应用程序时，必须先设置 HADOOP_CONF_DIR 环境变量。设置为 Hadoop 配置文件目录/usr/local/hadoop/etc/hadoop。

可以在“终端”程序中输入下列命令：

命令	说明
sudo gedit ~/.bashrc	编辑~/.bashrc

按下 Enter 键之后，就会以 gedit 打开~/.bashrc，屏幕显示界面如图 10-63 所示：

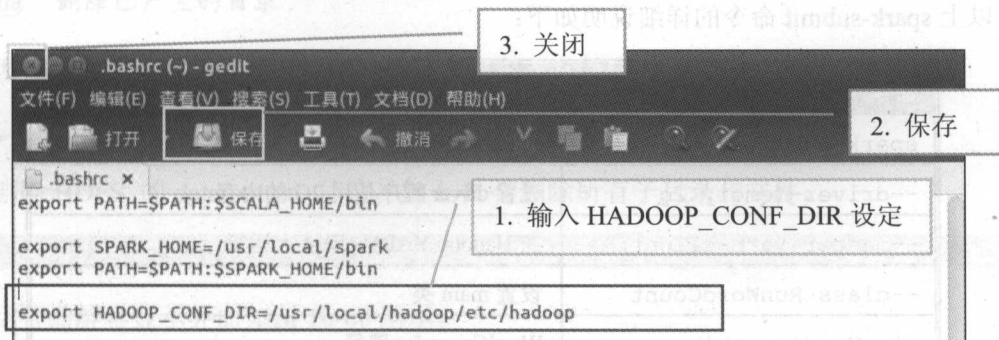


图 10-63 修改环境配置文件

步骤 05 让 ~/.bashrc 修改生效

可以从系统注销再重新登录或使用 `source ~/.bashrc` 命令，让用户的设置生效。

➤ 让用户环境变量的设置生效

```
source ~/.bashrc
```

运行后屏幕显示界面如图 10-64 所示。

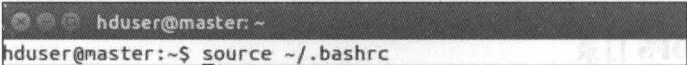


图 10-64 让 ~/.bashrc 修改生效

步骤 06 在 Hadoop YARN 上运行 WordCount 程序

在“终端”程序中输入下列命令，在 Hadoop YARN 上运行 WordCount：

➤ 切换至 WordCount 项目目录

```
cd ~/workspace/WordCount
```

➤ 在 Hadoop YARN 上运行 WordCount

```
spark-submit --driver-memory 2g --class RunWordCount --master yarn-client bin/WordCount.jar
```

运行后屏幕显示界面如图 10-65 所示。

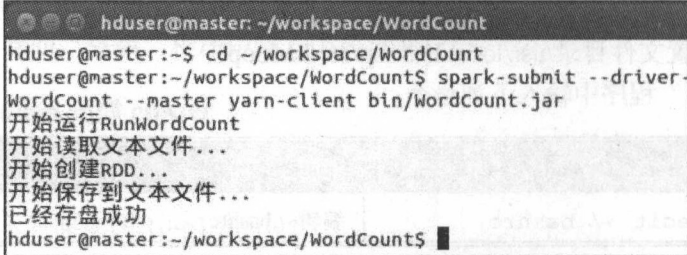


图 10-65 在 Hadoop YARN 上运行 WordCount 程序

以上 `spark-submit` 命令的详细说明如下：

命令	说明
spark-submit	spark-submit 命令
--driver-memory 2g	设置 driver 程序使用 2G 的内存
--master yarn-client	在 Hadoop YARN client 上运行
--class RunWordCount	设置 main 类
bin/WordCount.jar	WordCount Jar 路径

步骤 07 查看执行完成后 HDFS 产生的目录

运行后，输出文件产生在 HDFS data/output 目录中。在“终端”程序中输入下列命令：

命令	说明
<code>hadoop fs -ls data/output</code>	查看 HDFS 的 data/output 目录

运行后屏幕显示界面如图 10-66 所示。

```
hduser@master: ~/workspace/WordCount
hduser@master:~/workspace/WordCount$ hadoop fs -ls data/output
Found 3 items
-rw-r--r--  3 hduser supergroup          0 2016-05-16 23:23 data/output/_SUCCESS
-rw-r--r--  3 hduser supergroup    4400 2016-05-16 23:23 data/output/part-00000
-rw-r--r--  3 hduser supergroup    5124 2016-05-16 23:23 data/output/part-00001
hduser@master:~/workspace/WordCount$
```

图 10-66 查看运行完成后 HDFS 产生的目录

步骤 08 查看运行完成后 HDFS 产生的文件

在“终端”程序中输入下列命令，查看输出文件的内容。查看 HDFS 的 data/output/part-00000 文件：

```
hadoop fs -cat data/output /user/hduser/data/output/part-00000|more
```

运行后屏幕显示界面如图 10-67 所示。

```
hduser@master: ~/workspace/WordCount
hduser@master:~/workspace/WordCount$ hadoop fs -cat /user/hduser/data/output/part-00000|more
(intimately,1)
(Ahi,7)
(bone,15)
(Prophecies,1)
(park,,1)
(unfaithful,,1)
(riferire,,1)
(MALASPINA,,1)
(Mantua,1)
(derniers,1)
(hen,1)
```

图 10-67 查看运行完成后 HDFS 产生的文件

步骤 09 删除已产生的目录

因为我们后续还要继续测试，在“终端”程序中输入下列命令：

➤ 删除输出目录

删除 HDFS 的 data/output 目录，加上-R 会删除所有子目录与文件。

```
hadoop fs -rm -R data/output
```

运行后屏幕显示界面如图 10-68 所示。

```
hduser@master: ~/workspace/WordCount
hduser@master:~/workspace/WordCount$ hadoop fs -rm -R data/output
15/06/05 18:44:12 INFO fs.TrashPolicyDefault: Namenode trash configuration: Deletion interval = 0 minutes, Empty interval = 0 minutes.
Deleted data/output
```

图 10-68 删除已产生的目录

10.15 在 Spark Standalone Cluster 上运行 WordCount 程序

接下来要介绍如何使用 spark-submit 在 Standalone Cluster 上运行 WordCount Spark 程序。如果你尚未构建 Spark Standalone Cluster, 参考第 8.9 节来构建 Spark Standalone Cluster 运行环境。

步骤 01 启动 Standalone Cluster

在“终端”程序中输入下列命令:

➤ 启动 Standalone Cluster

```
/usr/local/spark/sbin/start-all.sh
```

运行后屏幕显示界面如图 10-69 所示, 从中可以看到一共启动了 6 个 worker。

```
hduser@master: ~/workspace/WordCount
hduser@master:~/workspace/WordCount$ /usr/local/spark/sbin/start-all.sh
starting org.apache.spark.deploy.master.Master, logging to /usr/local/spark/sbin/../logs/spark-hduser-org.apache.spark.deploy.master.Master-1-master.out
data2: starting org.apache.spark.deploy.worker.Worker, logging to /usr/local/spark/sbin/../logs/spark-hduser-org.apache.spark.deploy.worker.Worker-1-data2.out
data3: starting org.apache.spark.deploy.worker.Worker, logging to /usr/local/spark/sbin/../logs/spark-hduser-org.apache.spark.deploy.worker.Worker-1-data3.out
data1: starting org.apache.spark.deploy.worker.Worker, logging to /usr/local/spark/sbin/../logs/spark-hduser-org.apache.spark.deploy.worker.Worker-1-data1.out
data3: starting org.apache.spark.deploy.worker.Worker, logging to /usr/local/spark/sbin/../logs/spark-hduser-org.apache.spark.deploy.worker.Worker-2-data3.out
data2: starting org.apache.spark.deploy.worker.Worker, logging to /usr/local/spark/sbin/../logs/spark-hduser-org.apache.spark.deploy.worker.Worker-2-data2.out
data1: starting org.apache.spark.deploy.worker.Worker, logging to /usr/local/spark/sbin/../logs/spark-hduser-org.apache.spark.deploy.worker.Worker-2-data1.out
```

图 10-69 启动 Standalone Cluster

步骤 02 在 Spark Standalone Cluster 上运行 WordCount 程序

在“终端”程序中输入下列命令, 就可以在 Spark Standalone Cluster 运行 WordCount 程序:

➤ 切换至 WordCount 项目目录

```
cd ~/workspace/WordCount
```

➤ 在 Spark Standalone Cluster 上运行 WordCount 程序

```
spark-submit --driver-memory 2g --class RunWordCount --master spark://master:7077 bin/WordCount.jar
```


运行后屏幕显示界面如图 10-70 所示。

```
hduser@master: ~/workspace/WordCount
hduser@master:~$ cd ~/workspace/WordCount
hduser@master:~/workspace/WordCount$ spark-submit --driver-memory 2g --class Run
nWordCount --master spark://master:7077 bin/WordCount.jar
开始运行RunWordCount
开始读取文本文件...
开始创建RDD...
开始保存到文本文件...
已经存盘成功
hduser@master:~/workspace/WordCount$
```

图 10-70 在 Spark Standalone Cluster 上运行 WordCount 程序

以上 spark-submit 命令的详细说明如下：

命令	说明
spark-submit	spark-submit 命令
--driver-memory 2g	设置 driver 程序使用 2G 的内存
--master spark://master:7077	在 Spark Standalone Cluster 上运行
--class RunWordCount	设置 main 类
bin/WordCount.jar	WordCount Jar 路径

步骤 03 查看程序运行后的输出目录

在“终端”程序中输入下列命令：

➤ 查看产生在 HDFS data/output 目录中的输出文件

```
hadoop fs -ls data/output
```

运行后屏幕显示界面如图 10-71 所示。

```
hduser@master:~
hduser@master:~$ hadoop fs -ls data/output
Found 3 items
-rw-r--r-- 3 hduser supergroup 0 2015-06-05 19:09 data/output/_SUCCESS
-rw-r--r-- 3 hduser supergroup 202411 2015-06-05 19:09 data/output/part-00000
-rw-r--r-- 3 hduser supergroup 201166 2015-06-05 19:09 data/output/part-00001
```

图 10-71 查看产生在 HDFS data/output 目录中的输出文件

10.16 本书范例程序的安装说明

本书范例程序安装使用之前，必须构建整个开发环境。建议按照本书的顺序阅读第 1~10 章，并且实际演练构建整个开发环境后，再安装本书的范例程序。

步骤 01 下载范例程序

启动 master 服务器中的“终端”程序，再输入下列命令：

➤ 切换至用户 home 目录

```
cd ~/
```

如果登录的用户是 hduser，那么 home 目录就是/home/hduser。

➤ 下载范例程序

```
wget http://www.drmaster.com.tw/download/example/hadoopSparkExample.zip (或看本书提供的下载文件压缩包)
```

步骤 02 解压缩范例程序

然后在“终端”程序中输入下列命令，解压缩范例程序：

命令	说明
ll HadoopSparkExample.zip	查看下载的程序
unzip HadoopSparkExample.zip	解压缩范例程序

步骤 03 查看范例程序目录

解压缩后会产生两个目录：wordcount 与 SparkExample。

➤ wordcount 目录

在“终端”程序中输入下列命令来查看 wordcount 目录。

命令	说明
ll wordcount	查看第 7 章 WordCount.Java 范例程序。

运行后屏幕显示界面如图 10-72 所示。

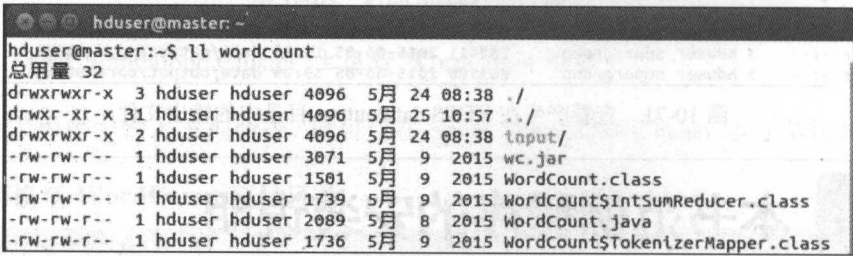


图 10-72 查看范例程序 wordcount 目录

以上程序是第 7 章 WordCount.Java 范例程序，示范使用 Hadoop MapReduce 计算文章内的每一个单词出现的次数。运行方式请自行参考第 7 章的说明。

➤ 查看 SparkExample 目录

在“终端”程序中输入下列命令。

```
ll SparkExample
```

执行后屏幕显示界面如图 10-73 所示：

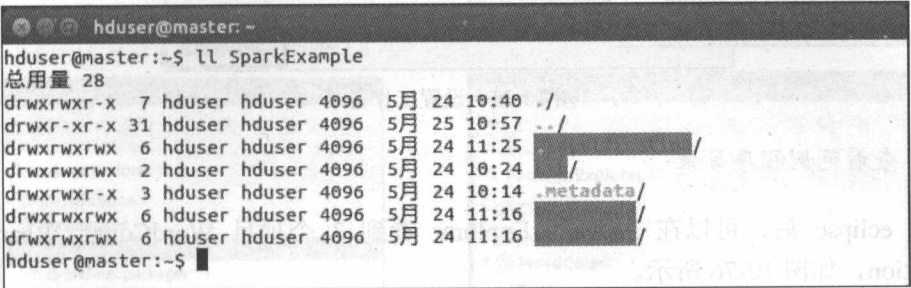


图 10-73 查看 SparkExample 目录

以上目录说明如下：

目录	说明
WordCount/	第 10 章范例
Recommend/	第 11 章范例
Classification/	第 12~18 章范例；分类算法

这些范例程序请自行参考相关章节的说明。

步骤 04 启动 eclipse（见图 10-74）

接下来，将使用 eclipse 打开 SparkExample 目录。



图 10-74 启动 eclipse

步骤 05 设置工作目录（见图 10-75）

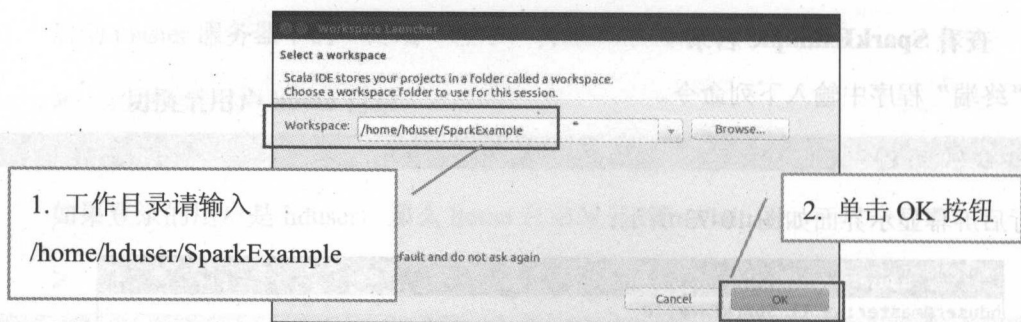


图 10-75 设置工作目录

步骤 06 查看范例程序目录

启动 eclipse 后, 可以在 Package Explorer 看到 3 个项目 WordCount、Recommend、Classification, 如图 10-76 所示。

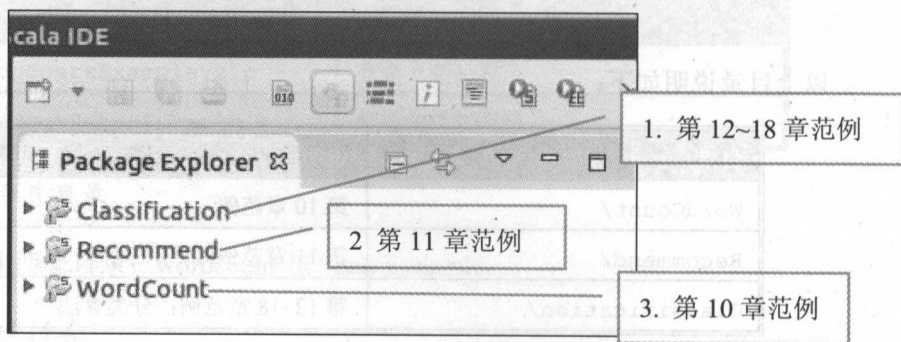


图 10-76 查看范例程序目录

步骤 07 查看 Classification 项目

可按照下列步骤 (见图 10-77) 查看 Classification 项目, 从中可以看到有关分类算法的相关程序。第 12~18 章会详细介绍。

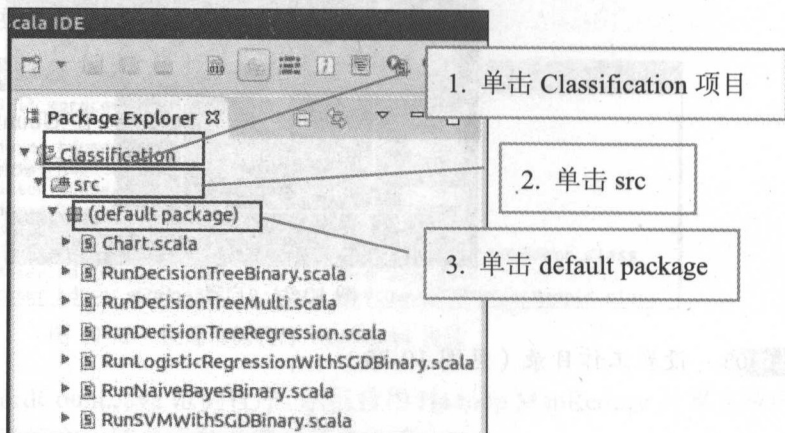


图 10-77 查看 Classification 项目

步骤 08 查看 Recommend 项目

可以按照相同步骤查看 Recommend 项目，如图 10-78 所示。推荐算法的相关程序，第 11 章会详细介绍。

步骤 09 查看 WordCount 项目

可以按照相同步骤查看 WordCount 项目，如图 10-79 所示，参考第 10 章的介绍。

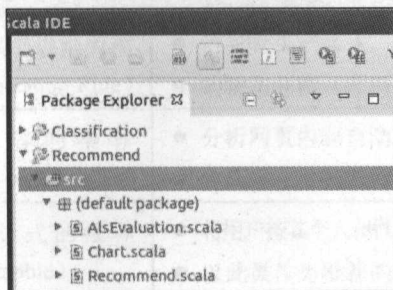


图 10-78 查看 Recommend 项目

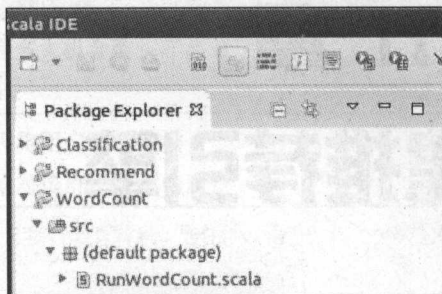


图 10-79 查看 WordCount 项目

步骤 10 范例程序编译与运行

有关范例程序的编译与运行，参考第 10.9 节的说明。如果安装的范例程序的工作目录不是 /home/hduser/SparkExample，可能会发生找不到链接库的情况，必须重新设置项目的链接库。参考第 10.5 节的说明。

第 11 章

创建推荐引擎

- 11.1 推荐算法介绍
- 11.2 “推荐引擎”大数据分析使用场景
- 11.3 ALS 推荐算法的介绍
- 11.4 ml-100k 推荐数据的下载与介绍
- 11.5 使用 spark-shell 导入 ml-100k 数据
- 11.6 查看导入的数据
- 11.7 使用 ALS.train 进行训练
- 11.8 使用模型进行推荐
- 11.9 显示推荐的电影名称
- 11.10 创建 Recommend 项目
- 11.11 Recommend.scala 程序代码
- 11.12 创建 PrepareData()数据准备
- 11.13 recommend()推荐程序代码
- 11.14 运行 Recommend.scala
- 11.15 创建 AlsEvaluation.scala 调校推荐引擎参数
- 11.16 创建 PrepareData()数据准备
- 11.17 进行训练评估
- 11.18 运行 AlsEvaluation
- 11.19 修改 Recommend.scala 为最佳参数组合

推荐引擎是最常见的机器学习应用，我们可以在各大购物网站上看见这方面的应用。

11.1 推荐算法介绍

常见推荐算法如下：

算法	说明
基于关系型规则的推荐 (Association Rule)	<ul style="list-style-type: none">● 消费者买产品 A，那么他有多大机会购买产品 B● 购物车分析（啤酒和尿布）
基于内容的推荐 (Content-based)	<ul style="list-style-type: none">● 分析网页内容自动分类，再将用户自动分类● 将新进已分类的网页推荐给对该群感兴趣的用户
人口统计式的推荐 (Demographic)	<ul style="list-style-type: none">● 将用户按其个人属性（性别、年龄、教育背景、居住地、语言）作为分类的指针● 以此类作为推荐的基础
协同过滤式的推荐 (Collaborative Filtering)	<ul style="list-style-type: none">● 通过观察到的所有用户对产品的评分，来推断用户的喜好● 找出对产品的评分相近的其他用户，他们喜欢的产品当前用户多半也会喜欢

本章主要是介绍协同过滤式的推荐，优缺点如下：

优点:	缺点:
<ul style="list-style-type: none">● 可以达到个性化推荐● 不需内容分析● 可以发现用户新的兴趣点● 自动化程度高	<ul style="list-style-type: none">● 冷启动问题 (Cold-start)：如果没有历史数据就没办法分析● 新用户问题：新用户没有评分，就不知道他的喜好

具体实现上多半是使用混合式方法。而结合上述的方法可以达到互补效果，进而提供用户更好的个性化推荐体验。

11.2 “推荐引擎”大数据分析使用场景

假设有一个 MoviesOnLine 的在线电影网站，会员可以付费在线观赏电影。公司希望能运用大数据分析推荐引擎，增加会员观看影片的次数以增加营收。因此，找来了大数据分析师组成一个团队，负责“电影推荐引擎”大数据项目。

➤ 找出问题

“问对问题”是解决问题的第一步。大数据分析师与公司人员讨论后发现，现有公司的推荐系统是人口统计式的推荐，必须具有个人属性数据。基于隐私权的原因，越来越难搜集到正确的个人属性数据。而且相同属性的用户未必会有相同的喜好，无法做到个性化的推荐。

➤ 设计解决方案模型

大数据分析师与公司人员讨论后，决定将推荐引擎加入协同过滤式的推荐，这种方法是通过对观察到所有会员给影片的评分来推断每个会员的喜好，并向会员推荐适合的影片。也就是说，和你观看影片喜好相近的用户，他喜欢的电影你多半也会喜欢。通过这种方式，可以达到个性化的推荐效果。

➤ 搜集数据

协同过滤式的推荐，最大的缺点是冷启动问题，如果没有历史数据就没办法分析推荐。还好因为网站已经运营了一段时间，累积了很多会员电影评分的数据和观看记录，可以使用这些数据构建协同过滤式的推荐引擎。

➤ 创建模型

大数据分析师决定使用 Spark MLlib 的 ALS (Alternating Least Squares) 推荐算法，采用这种方式可以解决稀疏矩阵 (Sparse Matrix) 的问题。即使是大量的用户与产品，都能够在合理的时间内完成运算。在使用历史数据训练后，就可以创建模型。

➤ 进行推荐

有了模型之后，就可以使用模型进行推荐。我们设计了如下推荐功能，可以增加会员观看电影的次数：

- **针对用户推荐感兴趣的电影：**可以针对每一位会员，定期发送短信或 E-mail 或会员登录时，推荐给他/她可能会感兴趣的电影。
- **针对电影推荐给感兴趣的会员：**当想要促销某些电影时，也可以找出可能会对这部电影感兴趣的会员，并且发送短信或 E-mail。

11.3 ALS 推荐算法的介绍

在 Spark MLlib 支持 ALS 推荐算法，是机器学习的协同过滤式推荐算法。机器学习的协同过滤式推荐算法通过观察所有用户给产品的评分，来推断每个用户的喜好，并向用户推荐适合的产品。

步骤 01 用户对产品项目的评分

用户对产品项目的评分有两种方式。现以只有 5 个用户与 5 个项目为例，说明如下：

显式评分 (Explicit Rating)

网站上的设计经常会请用户对某个产品进行评分，例如评分 1~5 颗星。可将其整理如下列矩阵：

	Item A	Item B	Item C	Item D	Item E
User 1	2	1	5		
User 2	1	3	1	1	
User 3	3			4	
User 4	2		2	1	2
User 5	1	1	1	4	1

隐式评分 (Implicit rating)

有时在网站的设计上,并不会请用户对某个产品进行评分,但是会记录用户是否点选了某个产品。如果点选了某个产品,代表该用户可能对该产品感兴趣,但是我们不知道评分为几颗星,这种方式称为隐式评分; 1 代表用户对该项产品有兴趣。

	Item A	Item B	Item C	Item D	Item E
User 1	1	1	1		
User 2	1	1	1	1	
User 3	1			1	
User 4	1		1	1	1
User 5	1	1	1	1	1

推荐算法就是找出两个用户喜好的相似性。例如 User 1 有兴趣的项目(A、B、C), User 2 有兴趣的项目(A、B、C、D)。User1 只比 User2 少了一个喜好项目 D。因此,当推荐算法要推荐项目给 User 1 时,会推荐项目 D。

步骤 02 稀疏矩阵的问题

如图 11-1 所示,当用户与项目评分越来越多时,会发现大部分都是空白的,这称为稀疏矩阵。

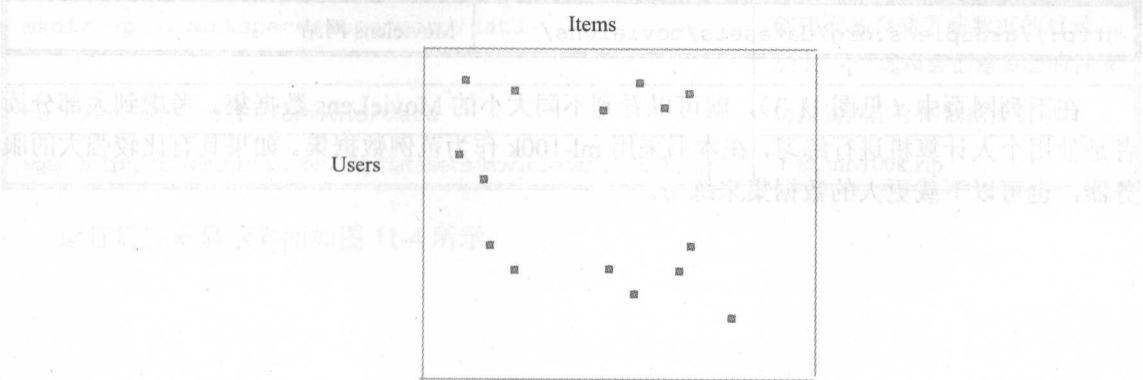


图 11-1 稀疏矩阵

用户与产品项目评分成千上万时，整个矩阵就会很大，而且很多都是空白。使用计算机来处理这样的矩阵很浪费内存，而且处理需要花费很多时间。

步骤 03 矩阵分解 (Matrix Factorization)

为了解决稀疏矩阵的问题，需采用矩阵分解。
如图 11-2 所示：将原本矩阵 $A(m \times n)$ 分解成 $X(m \times \text{rank})$ 矩阵与 $Y(\text{rank} \times n)$ 矩阵，而且 A 大约等于 $\approx X \times Y$ 。

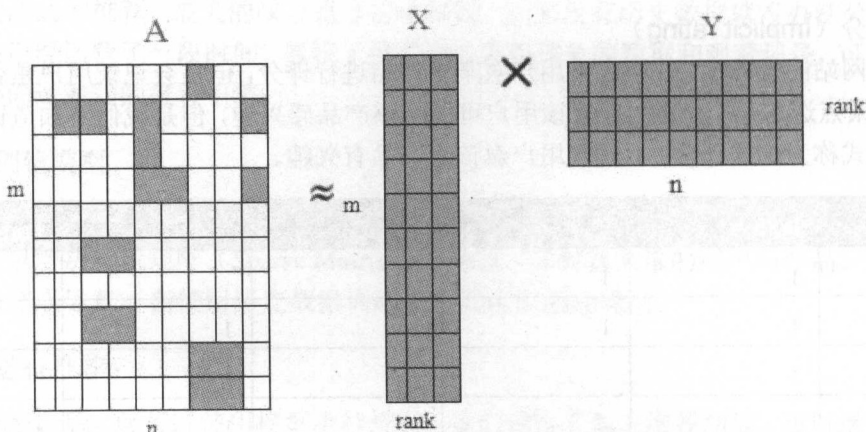


图 11-2 矩阵分解

11.4 ml-100k 推荐数据的下载与介绍

MovieLens 是一个推荐系统和虚拟社区网站，主要功能是使用协同过滤技术向会员推荐电影。GroupLens Research 实验室隶属于明尼苏达大学，MovieLens 网站是其中一个项目。在浏览器输入下列网址进入 MovieLens 网站：

网址	说明
http://grouplens.org/datasets/movielens/	MovieLens 网站

在下列网页中（见图 11-3），就可以看到不同大小的 MovieLens 数据集。考虑到大部分读者是使用个人计算机进行练习，在本书采用 ml-100k 作为范例数据集。如果具有比较强大的服务器，也可以下载更大的数据集来练习。

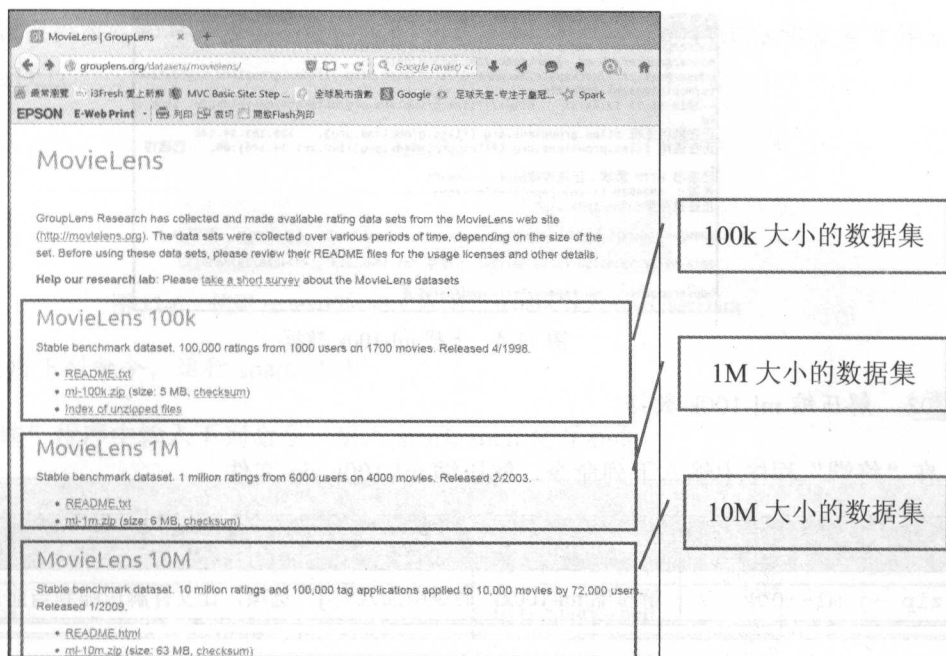


图 11-3 根据自己计算机的计算能力来选择不同大小的数据集来练习

本章使用的命令已经整理在本书的博客文章中，练习安装时可以复制博客文章中的命令，然后粘贴到“终端”程序中。这样可节省你打字的时间，也不用担心打错字（如果你无法在 VirtualBox 虚拟机的 Ubuntu “终端”程序中执行复制/粘贴操作时，参考第 3.9 节的说明，设置好 VirtualBox 的共享剪贴板）。本书的博客网址为：

<http://blog.sina.com.cn/hadoopsparkbook>

步骤 01 下载 ml-100k 数据

在“终端”程序中输入下列命令，下载 ml-100k 数据

命令	说明
<code>mkdir -p ~/workspace/Recommend/data</code>	创建准备存储下载数据的目录，加上“-p”选项会创建多层的目录
<code>cd ~/workspace/Recommend/data</code>	切换到预定存储数据的目录
<code>wget http://files.grouplens.org/datasets/movielens/ml-100k.zip</code>	下载 ml-100k.zip

运行后屏幕显示界面如图 11-4 所示。

```
hduser@master: ~/workspace/Recommend/data
hduser@master:~$ mkdir -p ~/workspace/Recommend/data
hduser@master:~$ cd ~/workspace/Recommend/data
hduser@master:~/workspace/Recommend/data$ wget http://files.grouplens.org/datasets/movielens/ml-100k.zip
--2016-05-17 13:41:51-- http://files.grouplens.org/datasets/movielens/ml-100k.zip
正在解析主机 files.grouplens.org (files.grouplens.org)... 128.101.34.146
正在连接 files.grouplens.org (files.grouplens.org)[128.101.34.146]:80... 已连接
已发出 HTTP 请求, 正在等待回应... 200 OK
长度: 4924029 (4.7M) [application/zip]
正在保存至: "ml-100k.zip"
100%[=====] 4,924,029 1.40MB/s 用时 3.5s
2016-05-17 13:41:56 (1.33 MB/s) - 已保存 "ml-100k.zip" [4924029/4924029]
hduser@master:~/workspace/Recommend/data$ █
```

图 11-4 下载 ml-100k 数据

步骤 02 解压缩 ml-100k 数据

在“终端”程序中输入下列命令，解压缩 ml-100k.zip 文件。

命令	说明
<code>unzip -j ml-100k</code>	解压缩 ml-100k; 命令中加入“-j”选项，让文件解压缩到当前目录中

运行后屏幕显示界面如图 11-5 所示。

```
hduser@master: ~/workspace/Recommend/data
hduser@master:~/workspace/Recommend/data$ unzip -j ml-100k
Archive: ml-100k.zip
  inflating: albut.pl
  inflating: mku.sh
  inflating: README
  inflating: u.data
  inflating: u.genre
  inflating: u.info
  inflating: u.item
  inflating: u.occupation
  inflating: u.user
  inflating: ui.base
```

图 11-5 解压缩 ml-100k 数据

步骤 03 ml-100k 文件数据的说明

可以在“终端”程序中使用 ll 命令查看有哪些文件，本章主要会使用到两个数据文件。

数据文件名称	说明
u.data	用户评分数据： 包含 4 个字段：user id（用户 id）、item id（项目 id）、rating（评分）、timestamp（日期时间）
u.item	电影的数据： 具有很多个字段，本书主要使用前 2 个字段：movie id（电影 id）、movie title（电影片名）

11.5 使用 spark-shell 导入 ml-100k 数据

为了让大家更容易理解 ALS 算法，我们先使用 spark-shell 示范使用 spark-shell 具有交互

性的好处。可以看到命令运行后的结果。如图 11-6 所示，将使用 `sc.textFile` 将文本文件 `u.data` 导入 `rawUserData`。

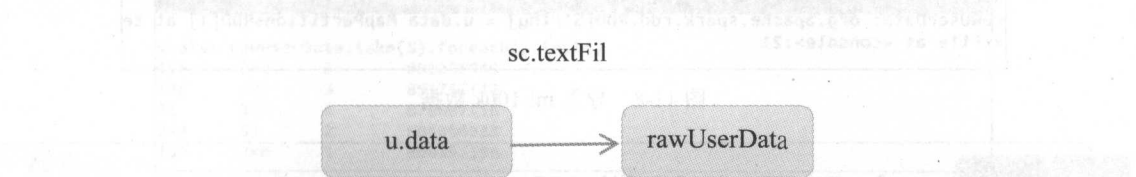


图 11-6 使用 `sc.textFile` 将文本文件 `u.data` 导入 `rawUserData`

步骤 01 使用下列命令，运行 `spark-shell`

在“终端”程序中输入下列命令，进入 `spark-shell` 交互界面。

命令	说明
<code>cd ~/workspace/Recommend/data</code>	进入 <code>data</code> 目录
<code>spark-shell</code>	运行 <code>Spark-shell</code>

运行完成后屏幕显示界面如图 11-7 所示，会看到 `scala` 提示符。

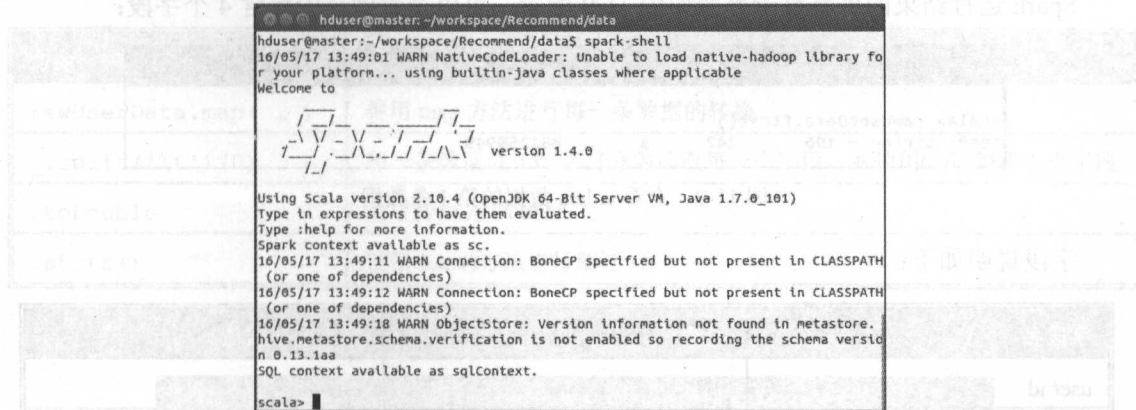


图 11-7 进入 `spark-shell` 交互界面

步骤 02 导入 `ml-100k` 数据

在 `spark-shell` 的 `scala>` 提示符下，输入下列命令导入 `ml-100k` 数据。

命令	说明
<code>val rawUserData = sc.textFile("u.data")</code>	将 <code>u.data</code> 数据使用 <code>sc.textFile</code> 导入 <code>rawUserData</code>

`Spark` 执行结果如图 11-8 所示。

```
hduser@master: ~/sparksamples/recommendation/ml-100k
scala> val rawUserData = sc.textFile("u.data")
rawUserData: org.apache.spark.rdd.RDD[String] = u.data MapPartitionsRDD[1] at te
xtFile at <console>:21
```

图 11-8 导入 ml-100k 数据

11.6 查看导入的数据

在使用数据进行训练之前可以先查看导入的数据，让你对数据特性有更多了解。

步骤 01 查看 u.data 数据的第 1 条数据

在 scala>提示符下，输入下列命令来查看 u.data 数据的第 1 条数据：

命令	说明
rawUserData.first()	查看第 1 条数据

Spark 运行结果的屏幕显示界面如图 11-9 所示，可以查看到其中包含 4 个字段：

```
hduser@master: ~/sparksamples/recommendation/ml-100k
scala> rawUserData.first()
res0: String = 196      242      3      881250949
```

图 11-9 查看 u.data 数据的第 1 条数据

字段说明如下：

字段名	说明	第 1 笔数据值
user id	用户 id	196
item id	项目 id	242
rating	评分	3
timestamp	日期时间	881250949

步骤 02 查看前 5 条数据

如果只看第 1 条数据不够，也可以使用下列 scala 语句显示多条数据：

命令	说明
rawUserData.take(5). foreach(println)	rawUserData 使用 take(5)读取前 5 条数据 .foreach(println)显示每一条数据

Spark 运行结果的屏幕显示界面如图 11-10 所示。

```
hduser@master: ~/sparksamples/recommendation/ml-100k
scala> rawUserData.take(5).foreach(println)
196    242    3    881250949
186    302    3    891717742
22     377    1    878887116
244    51     2    880606923
166    346    1    886397596
```

图 11-10 查看 u.data 数据的前 5 条数据

步骤 03 查看 userid（用户 id）字段的统计信息

在着手进一步处理数据之前，我们可以先使用下列命令大致了解每个字段数据的特性。我们可以使用下列命令，查看第 1 个字段:userid（用户 id）的统计信息。

命令	说明
<code>rawUserData.map(_._split('\t')(1).toDouble).stats()</code>	查看第 1 个字段: userid（用户 id）的统计信息

上述命令，详细说明如下：

命令	详细说明
<code>rawUserData.map(...)</code>	使用 map 方法进行每一条数据的转换
<code>_._split('\t')(0)</code>	每一条数据以 Tab 字符分隔读取每一个字段，并使用(0)取得第 1 个字段
<code>.toDouble</code>	文字转换为 Double
<code>.stats()</code>	使用 stats()方法进行统计

Spark 运行结果的屏幕显示界面如图 11-11 所示。

```
hduser@master: ~/sparksamples/recommendation/ml-100k
scala> rawUserData.map(_._split('\t')(0).toDouble).stats()
res2: org.apache.spark.util.StatCounter = (count: 100000, mean: 462.484750, stdev: 266.613087, max: 943.000000, min: 1.000000)
```

图 11-11 查看 userid（用户 id）字段的统计信息

说明如下：返回的数据类型是 org.apache.spark.util.StatCounter：

count（计数）：100000、mean（平均）：462.484750、stdev（标准偏差）：266.613087、max（最大值）：943.000000、min（最小值）：1.000000。

步骤 04 查看 item id（项目 id）字段的统计信息

我们可以使用下列命令查看第 2 个字段 item id 的统计信息：

命令	说明
<code>rawUserData.map(_._split('\t')(1).toDouble).stats()</code>	查看第 2 个字段 item id 的统计信息

上述命令详细说明如下：

命令	详细说明
<code>rawUserData.map(...)</code>	使用 map 方法进行每一条数据的转换
<code>_._split('\t')(1)</code>	每一条数据以 Tab 字符分隔读取每一个字段，并使用(1)读取第 2 个字段
<code>.toDouble</code>	文字转换为 Double
<code>.stats()</code>	使用 stats()方法进行统计

Spark 运行结果的屏幕显示界面如图 11-12 所示。

```
hduser@master: ~/sparksamples/recommendation/ml-100k
scala> rawUserData.map(_._split('\t')(1).toDouble).stats()
res3: org.apache.spark.util.StatCounter = (count: 100000, mean: 425.530130, stdev: 330.796702, max: 1682.000000, min: 1.000000)
```

图 11-12 查看 item id（项目 id）字段的统计信息

步骤 05 查看 rate（评分）字段的统计信息

我们可以使用下列命令查看第 3 个字段 rate（评分）的统计信息：

命令	说明
<code>rawUserData.map(_._split('\t')(2).toDouble).stats()</code>	查看第 3 个字段 rate（评分）的统计信息

上述命令的详细说明如下：

命令	详细说明
<code>rawUserData.map(...)</code>	使用 map 方法将每一条数据进行转换
<code>_._split('\t')(2)</code>	每一条数据以 Tab 字符分隔读取每一个字段，并使用(2)读取第 3 个字段
<code>.toDouble</code>	文字转换为 Double
<code>.stats()</code>	使用 stats()方法进行统计

Spark 运行结果的屏幕显示界面如图 11-13 所示。


```
hduser@master: ~/sparksamples/recommendation/ml-100k
scala> rawUserData.map(_._split('\t')(2).toDouble).stats()
res5: org.apache.spark.util.StatCounter = (count: 100000, mean: 3.529860, stdev: 1.125668, max: 5.000000, min: 1.000000)
```

图 11-13 查看 rate（评分）字段的统计信息

11.7 使用 ALS.train 进行训练

如图 11-14 所示，我们将使用 rawUserData 数据，以 map 转换为 rawRatings，再改用 map 转换为 ALS 训练数据格式 RDD[Rating]。然后使用 ALS.train 进行训练，训练完成后就会创建推荐引擎模型 MatrixFactorizationModel。

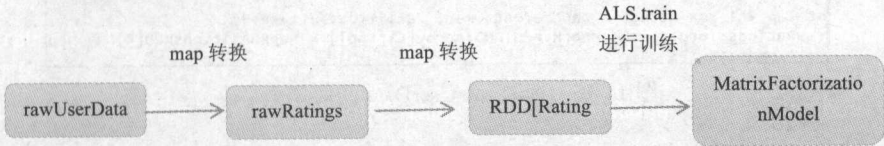


图 11-14 使用 ALS.train 进行训练的示意图

步骤 01 导入（Import）相关的 Lib

首先使用下列命令，导入链接库。

命令	说明
import org.apache.spark.mllib.recommendation.ALS	导入 ALS 链接库（library）
import org.apache.spark.mllib.recommendation.Rating	导入 Rating 链接库（library）

运行后的屏幕显示界面如图 11-15 所示。

```
hduser@master: ~
scala> import org.apache.spark.mllib.recommendation.ALS
import org.apache.spark.mllib.recommendation.ALS
scala> import org.apache.spark.mllib.recommendation.Rating
import org.apache.spark.mllib.recommendation.Rating
```

图 11-15 导入（Import）相关的 Lib

步骤 02 读取 rawUserData 的前 3 个字段

接下来我们要读取 rawUserData 的前 3 个字段，按照用户、产品、用户对此产品的评分来创建 rawRatings：

命令	说明
val rawRatings = rawUserData.map(_._split('\t') .take(3))	创建 rawRatings

上述命令的详细说明如下：

命令	说明
<code>val rawRatings = rawUserData.map(...)</code>	使用 <code>map</code> 方法针对 <code>rawUserData</code> 每一条数据进行转换，并将结果存入 <code>rawRatings</code>
<code>_.split('\t')</code>	每一条数据以 <code>split("\t")</code> <code>tab</code> 键分隔字段，读取每一个字段
<code>.take(3)</code>	读取前 3 个字段

运行后的屏幕显示界面如图 11-16 所示。

```
hduser@master: ~/workspace/Recommend/data
scala> val rawRatings = rawUserData.map(_.split("\t").take(3))
rawRatings: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[4] at map
at <console>:125
```

图 11-16 读取 rawUserData 的前 3 个字段

步骤 03 准备 ALS 训练数据

ALS 训练数据的格式是 `RDD[Rating]`数据类型，`Rating` 定义如下：

Rating(user: Int, product: Int, rating: Double) 数据类型		
字段	数据类型	说明
user	Int	用户
product	Int	产品
rating	Double	用户对此产品评分

可以使用下述命令创建 `ratingRDD`：

命令
<code>val ratingsRDD = rawRatings.map { case Array(user, movie, rating) =>Rating(user.toInt, movie.toInt, rating.toDouble) }</code>

上述命令的详细说明如下：

命令	详细说明
<code>val ratingsRDD = rawRatings.map { case Array(user, movie, rating)=> Rating(user.toInt, movie.toInt, rating.toDouble) }</code>	使用 <code>map</code> 方法针对 <code>rawRatings</code> 每一条数据进行转换，并将结果存入 <code>ratingsRDD</code> 这是 <code>case function</code> 传入 <code>Array(user, movie, rating)</code> 参数，并且返回 <code>Rating</code> 格式数据类型

运行完成后，屏幕显示界面如图 11-17 所示。

```
hduser@master: ~/workspace/Recommend/data
scala> val ratingsRDD = rawRatings.map { case Array(user, movie, rating) =>Rating(
user.toInt, movie.toInt, rating.toDouble) }
ratingsRDD: org.apache.spark.rdd.RDD[org.apache.spark.mllib.recommendation.Ratin
g] = MapPartitionsRDD[5] at map at <console>:27
```

图 11-17 准备 ALS 训练数据

步骤 04 使用 ALS.train 命令进行训练

我们将使用 ALS.train 命令进行训练，ALS.train 命令与参数说明如下。可分为显式评分训练与隐式评分训练：

显式评分训练		
ALS.train(ratings: RDD[Rating], rank: Int, iterations: Int, lambda: Double): MatrixFactorizationModel		
隐式评分训练		
ALS.trainImplicit (ratings: RDD[Rating], rank: Int, iterations: Int, lambda: Double): MatrixFactorizationModel		
说明	训练数据并返回模型	
参数	数据类型	说明
Ratings	RDD[Rating]	训练的数据格式是 Rating (userID, productID, rating) 的 RDD
rank	Int	rank 指的是当我们矩阵分解 Matrix Factorization 时，将原本矩阵 $A(m \times n)$ 分解成 $X(m \times \text{rank})$ 矩阵与 $Y(\text{rank} \times n)$ 矩阵
Iterations	Int	ALS 算法重复计算次数（建议值 10~20）
lambda	Double	建议值 0.01
返回数据	MatrixFactorizationModel	返回训练完成的模型，数据类型是 MatrixFactorizationModel

这些参数值的设置会影响结果的准确度，以及训练所需的时间。后续章节会说明如何调校找出最佳的参数组合。

训练完成后会产生 MatrixFactorizationModel 模型

训练时会执行矩阵分解，完成后会将原本矩阵 $A(m \times n)$ 分解成 $X(m \times \text{rank})$ 矩阵与 $Y(\text{rank} \times n)$ 矩阵，详细说明如下：

MatrixFactorizationModel 对象		
Member 成员	数据类型	说明
rank	Int	分解的参数
userFeatures	RDD[(Int, Array[Double])]	分解后用户矩阵 $X(m \times \text{rank})$
productFeatures	RDD[(Int, Array[Double])]	分解后产品矩阵 $Y(\text{rank} \times n)$

本范例使用显式评分 (Explicit rating) 训练。我们可以使用 ALS.train 命令，并且传入之前创建的 ratingsRDD 进行训练，将会返回 model 训练完成的模型：

命令	说明
val model = ALS.train(ratingsRDD, 10, 10, 0.01)	使用 ALS.train 显式评分(Explicit rating)训练，传入创建的 ratingsRDD 以及参数，训练完成后返回 model

运行后屏幕显示界面如图 11-18 所示。

```
hduser@master: ~/workspace/Recommend/data
scala> val model = ALS.train(ratingsRDD, 10, 10, 0.01)
15/06/16 23:08:52 WARN BLAS: Failed to load implementation from: com.github.fommil.netlib.NativeSystemBLAS
15/06/16 23:08:52 WARN BLAS: Failed to load implementation from: com.github.fommil.netlib.NativeSystemBLAS
15/06/16 23:08:53 WARN LAPACK: Failed to load implementation from: com.github.fommil.netlib.NativeSystemLAPACK
15/06/16 23:08:53 WARN LAPACK: Failed to load implementation from: com.github.fommil.netlib.NativeRefLAPACK
model: org.apache.spark.mllib.recommendation.MatrixFactorizationModel = org.apache.spark.mllib.recommendation.MatrixFactorizationModel@42ad4ad2
```

图 11-18 使用 ALS.train 命令进行训练

11.8 使用模型进行推荐

在前面章节中，我们已经训练完成模型，接下来将使用此模型进行推荐。

步骤 01 针对用户推荐电影

我们可以针对每一位会员，定期发送短信或 E-mail 或在会员登录时，向会员推荐可能会感兴趣的电影。

针对用户推荐电影，我们可以使用 model.recommendProducts 方法来推荐，说明如下：

MatrixFactorizationModel.recommendProducts(user: Int, num: Int): Array[Rating] 方法		
说明	输入参数 user，针对此 user 推荐给他/她有可能感兴趣的产品	
参数	数据类型	说明
user	Int	要被推荐的用户 id
num	Int	推荐的记录数
返回	Array[Rating]	返回 Rating 数据类型(user: Int, product: Int, rating: Double)的数组：数组中每一条都是系统的推荐产品，rating 是系统给的评分，rating 越高代表系统越加优先向此用户推荐此产品；返回的数组会按 rating 从大到小排序，也就是说第 1 条是系统推荐评分最高的产品

上列方法的使用范例，例如以下针对用户 196 推荐前 5 部电影：

命令	说明
<code>model.recommendProducts(196,5).mkString("\n")</code>	使用训练完成的模型进行推荐，传入参数(user=196,num=5)，mkString 将每一条数据加入\n 换行符号产生字符串

运行后屏幕显示界面如图 11-19 所示。

```
hduser@master: ~/workspace_book/Recommend/data
scala> model.recommendProducts(196,5).mkString("\n")
res2: String =
Rating(196,464,10.037872848767316)
Rating(196,1085,8.743486699327445)
Rating(196,798,8.677965118879271)
Rating(196,998,8.312921766939127)
Rating(196,902,7.84996286425107)
```

图 11-19 针对用户 196 推荐前 5 部电影

如图 11-19 所示，第一条数据是系统针对此用户首先推荐的产品：

Rating(196,464,10.037872848767316)

其意义是推荐给用户 ID 196，产品 ID 464，推荐评分 10.37872848767316。

推荐评分越高，则代表系统越优先推荐此产品。

步骤 02 查看针对用户推荐产品的评分

可以查询系统对用户推荐产品的评分，如下述命令所示，系统针对用户 196 推荐产品 464 的评分：

命令	说明
<code>model.predict(196, 464)</code>	系统针对用户 196 推荐产品 464 的评分

屏幕显示界面如图 11-20 所示，运行后返回 10.37872848767316：

```
hduser@master: ~/workspace_book/Recommend/data
scala> model.predict(196, 464)
res3: Double = 10.037872848767316
```

图 11-20 查看针对用户 196 推荐产品 464 的评分

步骤 03 针对电影推荐给用户

当我们想要促销某些电影时，可以找出可能会对这些电影感兴趣的会员，并且发送短信或 E-mail。

针对电影推荐给用户，我们可以使用 `model.recommendUsers` 方法推荐，说明如下：

MatrixFactorizationModel.recommendUsers(product: Int, num: Int): Array[Rating]方法		
说明	输入参数 product，针对此 product 推荐给可能有兴趣的用户	
参数	数据类型	说明
product	Int	要被推荐的产品 id
num	Int	推荐的记录数
返回	Array[Rating]	返回 Rating 数据类型(user: Int, product: Int, rating: Double)的数组： 数组中每一条都是系统针对产品推荐给用户，rating 是系统给的评分， rating 越高代表针对此产品越优先推荐给用户；返回的数组会按 rating 从大到小排序，也就是说第 1 条是系统推荐最有趣味的用户

可以使用下述命令，针对电影 464 推荐前 5 位用户：

命令	说明
<code>model.recommendUser(464,5).mkString("\n")</code>	使用训练完成的模型进行推荐，传入参数(user=196 ,num=5)；并且 针对每一条数据加入\n 换行符号产生字符串

运行后屏幕显示界面如图 11-21 所示。

```
hduser@master: ~/workspace/Recommend/data
scala> model.recommendUsers(464,5).mkString("\n")
res1: String =
Rating(78,464,10.623067359174735)
Rating(9,464,8.902707270368083)
Rating(98,464,8.570439840792409)
Rating(93,464,8.440980018938644)
Rating(212,464,8.05463415520267)
```

图 11-21 针对电影 464 推荐给前 5 位用户

如图 11-21 所示，第一条数据是系统针对电影推荐给用户：

Rating(78,464,10.623067359174735)

其意义是针对电影 ID 464 推荐给用户 ID 78，推荐评分为 10.623067359174735。

11.9 显示推荐的电影名称

之前的范例只显示推荐电影的 ID，接下来将介绍如何显示推荐电影的名称：

步骤 01 创建电影 ID 与名称的对照表

为了显示推荐电影的名称，必须创建电影 ID 与名称的对照表，使用下述命令创建对照表：

```
val itemRDD = sc.textFile("u.item")
```

```
val movieTitle= itemRDD.map(line => line.split("\\|").take(2)).map(array=>
(array(0).toInt,array(1))).collectAsMap()
```

上述命令说明如下：

命令	说明
val itemRDD = sc.textFile("u.item")	使用 sc.textFile 将 u.item 文本文件导入 itemRDD
val movieTitle= itemRDD	itemRDD 经过一系列命令计算，将结果存入 movieTitle 对照表
.map(line => line.split("\\ ").take(2))	使用 map 方法针对每一条数据进行转换： 每一条数据使用 line.split("\\ ") 以 符号分隔 字段，并以.take(2)读得前 2 个字段
.map(array=> (array(0).toInt,array(1)))	使用 map 方法将上一命令读取的前 2 个字段进 行转换；array(0).toInt 将第 1 个字段转换为 Int， 也就是电影 ID；array(1)是第 2 个字段，为电影 的名称
.collectAsMap()	使用 collectAsMap()，创建 movieTitle 电影 ID / 名称对照表

运行完成后屏幕显示界面如图 11-22 所示。

```
hduser@master: ~/workspace_book/Recommend/data
scala> val itemRDD = sc.textFile("u.item")
itemRDD: org.apache.spark.rdd.RDD[String] = u.item MapPartitionsRDD[218] at text
File at <console>:23

scala> val movieTitle= itemRDD.map(line => line.split("\\|").take(2)).map(array=
> (array(0).toInt,array(1))).collectAsMap()
movieTitle: scala.collection.Map[Int,String] = Map(137 -> Big Night (1996), 891
-> Bent (1997), 550 -> Die Hard: With a Vengeance (1995), 1205 -> Secret Agent,
The (1996), 146 -> Unhook the Stars (1996), 864 -> My Fellow Americans (1996), 5
59 -> Interview with the Vampire (1994), 218 -> Cape Fear (1991), 568 -> Speed (
1994), 227 -> Star Trek VI: The Undiscovered Country (1991), 765 -> Boomerang (1
992), 1115 -> Twelfth Night (1996), 774 -> Prophecy, The (1995), 433 -> Heathers
(1989), 92 -> True Romance (1993), 1528 -> Nowhere (1997), 846 -> To Gillian on
Her 37th Birthday (1996), 1187 -> Switchblade Sisters (1975), 1501 -> Prisoner
of the Mountains (Kavkazsky Plennik) (1996), 442 -> Amityville Curse, The (1990)
, 1160 -> Love! Valour! Compassion! (1997), 101 -> Heavy Metal (1981), 1196 -...
```

图 11-22 创建电影 ID 与名称的对照表

步骤 02 显示对照表的前 5 条数据

之前我们已经创建 movieTitle 对照表，现在可以使用下述命令查看对照表前 5 条数据：

命令	说明
movieTitle.take(5).foreach(println)	查询 movieTitle 对照表前 5 条数据

运行后屏幕显示界面如图 11-23 所示，例如可以看到第一条 MovieID 146 对应到电影名称 Unhook the Stars (1996)：

```
hduser@master: ~/workspace_book/Recommend/data

scala> movieTitle.take(5).foreach(println)
(146,Unhook the Stars (1996))
(1205,Secret Agent, The (1996))
(550,Die Hard: With a Vengeance (1995))
(891,Bent (1997))
(137,Btg Night (1996))
```

图 11-23 显示对照表的前 5 条数据

步骤 03 查询电影名称

有了对照表之后，可以使用下述命令查询每一个 movie ID 的电影名称：

命令	说明
movieTitle (146)	查询 movie ID 196 的电影名称

运行后屏幕显示界面如图 11-24 所示，显示电影名称 Unhook the Stars (1996)

```
hduser@master: ~/workspace/Recommend/data

scala> movieTitle(146)
res3: String = Unhook the Stars (1996)
```

图 11-24 查询电影名称

步骤 04 显示前 5 条推荐的电影名称

有了 movieTitle 对照表，我们就可以显示推荐的电影名称，例如下述命令：

```
model.recommendProducts(196,5).map(rating =>
(rating.product,movieTitle(rating.product), rating.rating)).foreach(println) ==>
```

上述命令的详细说明如下：

命令	说明
model.recommendProducts(196,5)	针对用户 196 推荐 5 条电影
.map(rating => (rating.product, movieTitle(rating.product), rating.rating)))	使用 map 进行每一条数据转换 产品 id（不须转换） 使用 movieTitle 对照表传入产品 id，并且转换为电影名称 评分（不须转换）
.foreach(println)	每一条数据输出

运行的结果界面如图 11-25 所示，我们可以看到已经显示了电影名称：


```

hduser@master: ~/workspace_book/Recommend/data
scala> model.recommendProducts(196,5).map(rating => (rating.product, movieTitle(rating.product), rating.rating)).foreach(println)
(464,Vanya on 42nd Street (1994),10.037872848767316)
(1085,Carried Away (1996),8.743486699327445)
(798,Bad Company (1995),8.677965118879271)
(998,Cabin Boy (1994),8.312921766939127)
(902,Big Lebowski, The (1998),7.84996286425107)

```

图 11-25 显示前 5 条推荐的电影名称

11.10 创建 Recommend 项目

在之前的章节中，我们已经使用 spark-shell 示范了如何导入数据、创建 Rating RDD、训练模型、推荐产品，对推荐算法流程已经有了基本概念。虽然使用 spark-shell 具有交互性的好处，输入命令立刻可以看到响应。然而缺点是无法重复使用。因此将介绍如何创建 Spark 应用程序可以来重复使用。

步骤 01 启动 eclipse (见图 11-26)

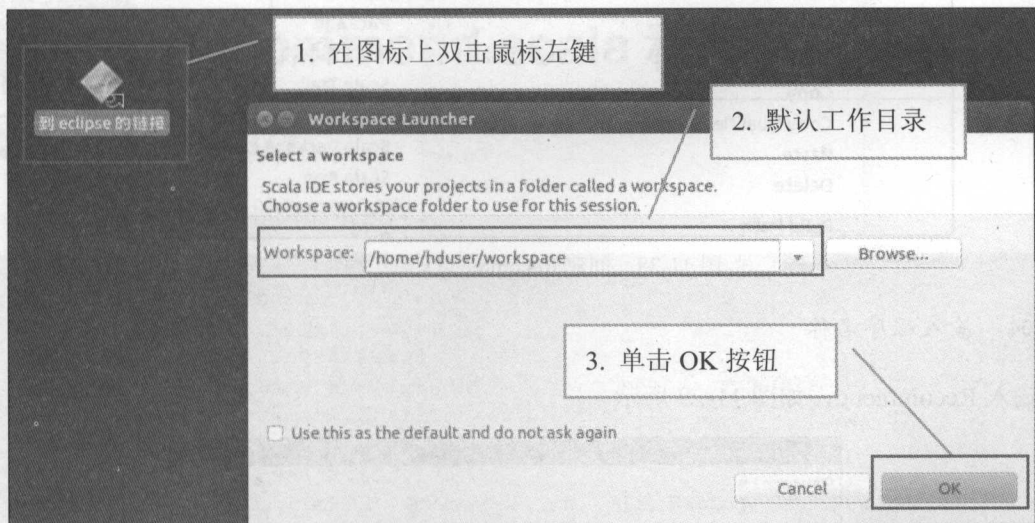


图 11-26 启动 eclipse

步骤 02 创建 Recommend 项目

参考第 10.4 节来创建新的 scala 项目，创建 Recommend 项目，完成后屏幕显示界面如图 11-27 所示。

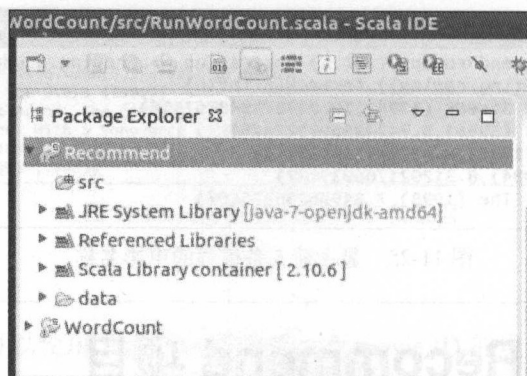


图 11-27 创建 Recommend 项目

步骤 03 创建 Recommend.scala 文件（见图 11-28）

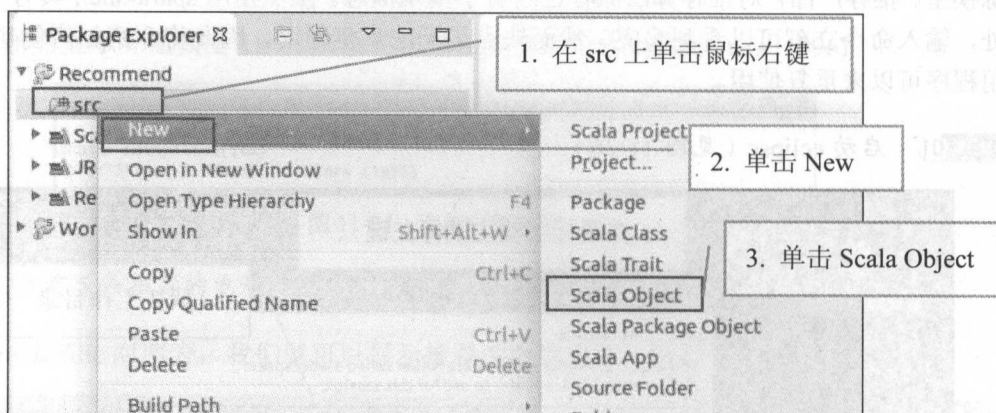


图 11-28 创建 Recommend.scala 文件

步骤 04 输入程序名称

输入 Recommend，如图 11-29 所示。

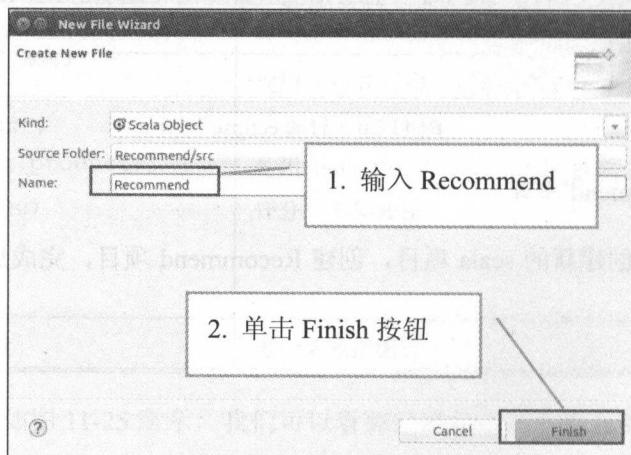


图 11-29 输入程序名称

步骤 05 创建完成的 Scala 程序（见图 11-30）

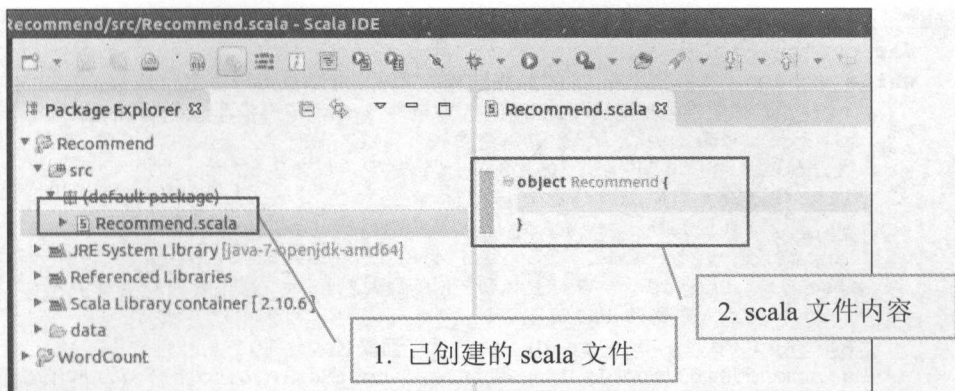


图 11-30 创建完成的 Scala 程序

步骤 06 创建 Recommend 项目

接下来，我们就要开始输入程序代码了。

11.11 Recommend.scala 程序代码

步骤 01 Import 导入链接库

先使用 Import 导入相关的链接库：

```
import java.io.File
import scala.io.Source
import org.apache.log4j.Logger
import org.apache.log4j.Level
import org.apache.spark.SparkConf
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.rdd._
import org.apache.spark.mllib.recommendation.{ ALS, Rating, MatrixFactorizationModel }
import scala.collection.immutable.Map
```

Scala import 语句说明如下：

语句	说明
import org.apache.spark.SparkContext._	加入 “_” 代表要导入 org.apache.spark.SparkContext 软件包内的所有类
import org.apache.spark.mllib. recommendation.{ ALS, Rating, MatrixFactorizationModel }	“{ }” 代表要导入 org.apache.spark.mllib.recommendation 软件包内的 ALS、Rating、MatrixFactorizationModel 类

步骤 02 main 主程序代码

创建 main function 程序代码。main function 是我们程序的起点：

```
def recommend(model: MatrixFactorizationModel, movieTitle: Map[Int, String]) = {
  var choose = ""
  while (choose != "3") { //如果选择3. 离开,就结束运行程序
    print("请选择要推荐类型 1.针对用户推荐电影 2.针对电影推荐给感兴趣的用户 3.离开?")
    choose = readLine() //读取用户输入
    if (choose == "1") { //如果输入1.针对用户推荐电影
      print("请输入用户 id?")
      val inputUserID = readLine() //读取用户 ID
      RecommendMovies(model, movieTitle, inputUserID.toInt) //针对此用户推荐电影
    } else if (choose == "2") { //如果输入2.针对电影推荐给感兴趣的用户
      print("请输入电影的 id?")
      val inputMovieID = readLine() //读取 MovieID
      RecommendUsers(model, movieTitle, inputMovieID.toInt) //针对此电影推荐用户
    }
  }
}
```

main 程序代码分为 3 部分，

程序代码	说明
val (ratings, movieTitle) = PrepareData()	数据准备阶段： 执行后会产生 ratings（评分数据）与 movieTitle（电影 ID / 名称对照表）
val model = ALS.train(ratings, 5, 20, 0.1)	训练阶段： 传入的 ratings（评分数据）与参数进行训练，并且产生训练模型
recommend(model, movieTitle)	推荐阶段： 传入训练模型与 movieTitle（电影 ID / 名称对照表）进行推荐

以上程序可以整理为如下架构图（见图 11-31）：

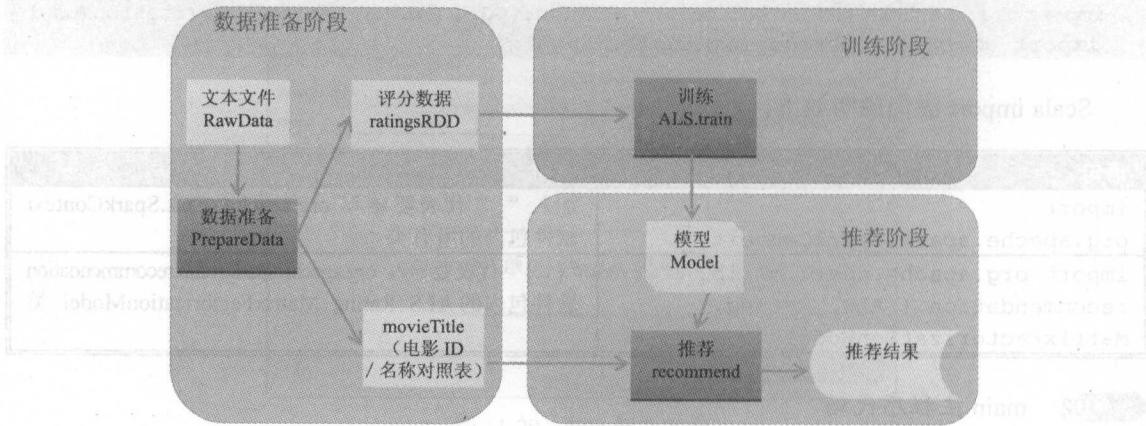


图 11-31 Recommend 程序的架构图

步骤 03 SetLogger 设置不显示 log 信息

因为 Spark 运行程序时会显示很多 log 信息，信息太多会和正常信息混合在一起，所以设置不显示 log 信息：

```
def SetLogger = {
  Logger.getLogger("org").setLevel(Level.OFF)
  Logger.getLogger("com").setLevel(Level.OFF)
  System.setProperty("spark.ui.showConsoleProgress", "false")
  Logger.getLogger().setLevel(Level.OFF);
}
```

11.12 创建 PrepareData()数据准备

步骤 01 创建 PrepareData()函数

数据准备主要分为 3 步骤，先输入注释行：

```
def PrepareData(): (RDD[Rating], Map[Int, String]) = {
  //-----1.创建用户评分数据-----
  //-----2.创建电影 ID 与名称对照表-----
  //-----3.显示数据记录数-----
}
```

PrepareData()数据准备步骤说明如下：

数据准备步骤	说明
1. 创建用户评分数据	读取 u.data 数据，转换为 ratingsRDD 作为后续训练数据
2. 创建电影 ID 与名称对照表	读取 u.item 数据，转换为 movieTitle 对照表
3. 显示数据记录数	显示用户与电影不重复记录数
返回	ratingsRDD 与 movieTitle

步骤 02 创建用户评分数据

首先创建用户评分数据 ratingsRDD，作为后续训练数据：

```
val sc = new SparkContext(new SparkConf().setAppName("Recommend").setMaster(
  "local[4]"))
print("开始读取用户评分数据中...")
val rawUserData = sc.textFile("file:/home/hduser/workspace/Recommend/data/
u.data")
val rawRatings = rawUserData.map(_.split("\t").take(3))
val ratingsRDD = rawRatings.map { case Array(user, movie, rating) =>
```

```
Rating(user.toInt, movie.toInt, rating.toDouble) }
println("共计: " + ratingsRDD.count.toString() + "条 ratings")
```

上列程序代码的详细说明如下:

程序代码	说明
val sc = new SparkContext(new SparkConf() .setAppName("Recommend").setMaster("local[4]"))	创建 SparkContext
val DataDir = "data"	设置数据文件所在的子目录名称
val rawUserData = sc.textFile(new File(DataDir, "u.data").toString)	读取 u.data 并载入 rawUserData
val rawRatings = rawUserData.map(_._split("\t"). take(3))	因为文本文件是以 tab 字符分隔字段, 所以使用"\t"分隔字符串并且读出前3个字段, 分别是 user、movie、rating
val ratingsRDD = rawRatings.map { case Array(user, movie, rating) => Rating(user.toInt, movie.toInt, rating. toDouble) }	rawRatings 使用 map 转换, 创建 Rating 数据类型 (userId, movieId, rating)

步骤 03 创建电影 ID 与名称对照表

接下来, 创建电影 ID 与名称对照表。

```
//-----2.创建电影 ID 与名称对照表-----
print("开始读取电影数据中...")
val itemRDD = sc.textFile(new File(DataDir, "u.item").toString)
val movieTitle = itemRDD.map(line => line.split("\\|").take(2))
  .map(array => (array(0).toInt, array(1))).collect().toMap
```

上列程序代码, 详细说明如下:

程序代码	说明
val itemRDD = sc.textFile(new File(DataDir, "u.item").toString)	读取 u.item 并载入 itemRDD
val movieTitle = itemRDD.map(line => line.split("\\ ").take(2))	因为文本文件是以 " " 符号分隔字段, 并且读取前 2 个字段, 分别是 movieID、movieTitle
.map(array => (array(0).toInt, array(1))).collect().toMap	使用 map 函数产生(movieId, movieName)对照表

步骤 04 显示数据记录数

接下来显示数据记录数。

```
//-----3.显示数据记录数-----
val numRatings = ratingsRDD.count()
val numUsers = ratingsRDD.map(_._user).distinct().count()
val numMovies = ratingsRDD.map(_._product).distinct().count()
println("共计: ratings: " + numRatings + " User " + numUsers + " Movie " +
numMovies)
return (ratingsRDD, movieTitle)
}
```

上列程序代码的详细说明如下：

程序代码	说明
val numRatings = ratingsRDD.count()	ratingRDD 数据记录数
val numUsers = ratingsRDD.map(_._user).distinct().count()	不重复用户数
val numMovies = ratingsRDD.map(_._product).distinct().count()	不重复产品共计数
println("共计: ratings: " + numRatings + " User " + numUsers + " Movie " + numMovies)	显示共计数
return (ratingsRDD, movieTitle)	返回 ratingsRDD 与 movieTitle 对照表

11.13 recommend()推荐程序代码

步骤 01 recommend()推荐程序代码

输入 recommend()推荐程序代码如下：

```
def recommend(model: MatrixFactorizationModel, movieTitle: Map[Int, String]) = {
  var choose = ""
  while (choose != "3") { //如果选择3.离开,就结束运行程序
    print("请选择要推荐类型 1.针对用户推荐电影; 2.针对电影推荐给感兴趣的用户; 3.离开?")
    choose = readLine() //读取用户输入
    if (choose == "1") { //如果输入1.针对用户推荐电影
      print("请输入用户 id?")
      val inputUserID = readLine() //读取用户 ID
      RecommendMovies(model, movieTitle, inputUserID.toInt) //针对此用户推荐电影
    } else if (choose == "2") { //如果输入2.针对电影推荐给感兴趣的用户
      print("请输入电影的 id?")
      val inputMovieID = readLine() //读取 MovieID
      RecommendUsers(model, movieTitle, inputMovieID.toInt) //针对此电影推荐用户
    }
  }
}
```


以上程序让用户选择推荐类型：1.针对用户推荐电影；2.针对电影推荐给感兴趣的用户；
3.离开。

说明如下：

选择	说明
选择 1	针对此用户推荐电影 RecommendMovies(model, movieTitle, inputUserID.toInt)
选择 2	针对此电影推荐用户 RecommendUsers(model, movieTitle, inputMovieID.toInt)
选择 3	结束运行程序

步骤 02 RecommendMovies()针对此用户推荐电影

针对此用户推荐电影：

```
def RecommendMovies(model: MatrixFactorizationModel, movieTitle: Map[Int, String], inputUserID: Int) = {  
    val RecommendMovie = model.recommendProducts(inputUserID, 10)  
    var i = 1  
    println("针对用户 id" + inputUserID + "推荐下列电影:")  
    RecommendMovie.foreach { r =>  
        println(i.toString() + "." + movieTitle(r.product) + " 评分：" +  
r.rating.toString())  
        i += 1  
    }  
}
```

以上程序说明如下：

程序代码	说明
def RecommendMovies(model: MatrixFactorizationModel, movieTitle: Map[Int, String], inputUserID: Int) = {	定义 RecommendMovies 函数参数： model 为训练完成的模型 movieTitle 为电影 ID 名称对照表 inputUserID 为输入的用户 ID
val RecommendMovie = model.recommendProducts(inputUserID, 10)	获取针对 inputUserID 推荐前 10 部电影，并存入 RecommendMovie
RecommendMovie.foreach { r => println(i.toString()+"."+movieTitle(r.product) +"评分:"+r.rating.toString()) i += 1 }	使用 foreach 读取 RecommendMovie 的每一条数据，并且显示在界面上

步骤 03 RecommendUsers()针对电影推荐给用户

针对电影推荐用户：

```
def RecommendUsers(model: MatrixFactorizationModel, movieTitle: Map[Int, String], inputMovieID: Int) = {
    val RecommendUser = model.recommendUsers(inputMovieID, 10) // 读取针对 inputMovieID 推荐前10位用户
    var i = 1
    println("针对电影 id" + inputMovieID + "电影名:" + movieTitle (inputMovieID.toInt) + "推荐下列用户 id:")
    RecommendUser.foreach { r =>
        println(i.toString + "用户 id:" + r.user + " 评分:" + r.rating)
        i = i + 1
    }
}
```

推荐用户主要使用 MatrixFactorizationModel.recommendUsers(product: Int, num: Int): Array[Rating]方法，说明如下：

程序代码	说明
def RecommendUsers(model: MatrixFactorizationModel, movieTitle: Map[Int, String], inputUserID: Int) = {	定义 RecommendMovies 函数参数： <ul style="list-style-type: none">● model 为训练完成的模型● movieTitle 为电影 ID 名称对照表● inputUserID 为输入的用户 ID
val RecommendUser = model.recommendUsers(inputMovieID, 10)	获取针对 inputMovieID 推荐前 10 位用户，并存入 RecommendUser
RecommendUser.foreach { r => println(i.toString + "用户 id:" + r.user + " 评分:" + r.rating) i = i + 1 }	读取 RecommendUser 每一条数据，并且输出在屏幕界面上

11.14 运行 Recommend.scala

程序已经输入完成，接下来要开始运行了。

步骤 01 依次选择菜单及其菜单项：Run→Run Configurations

运行程序前，必须先进行设置，如图 11-32 所示。

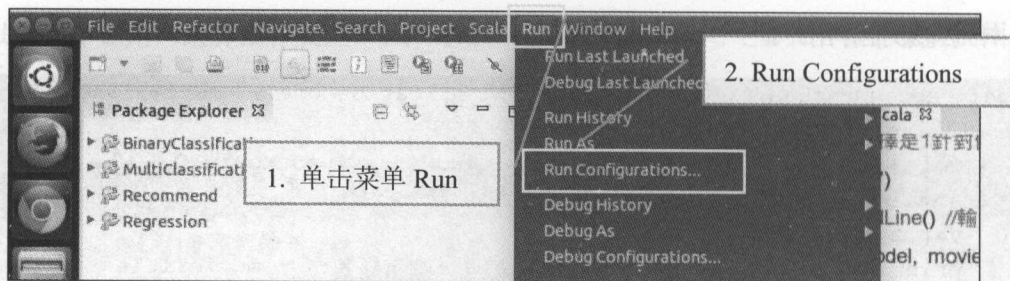


图 11-32 依次选择菜单及其菜单项：Run→Run Configurations

步骤 02 设置 Run Configurations (见图 11-33)

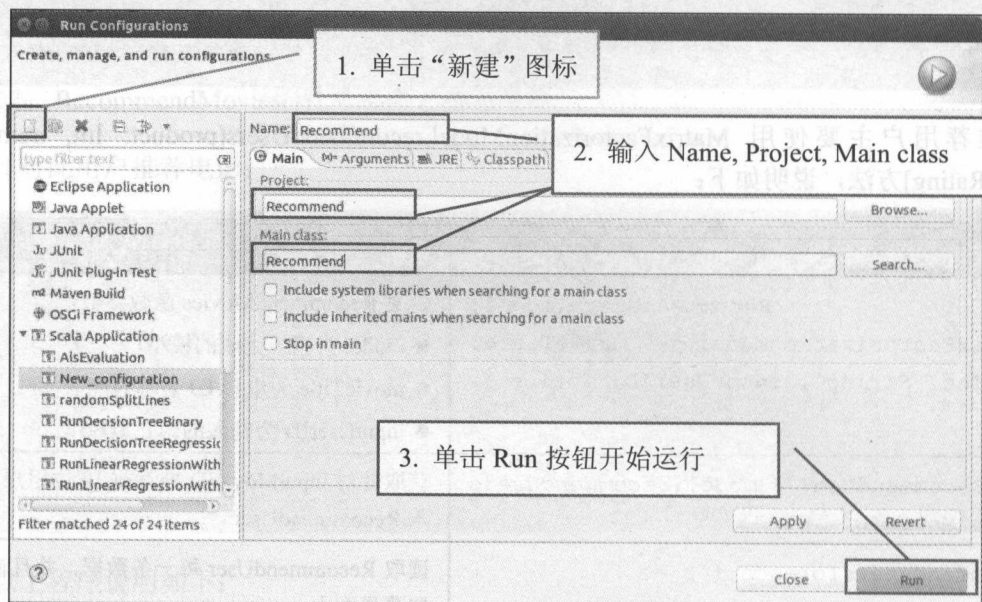


图 11-33 设置 Run Configurations

步骤 03 运行界面

运行之后，Console 出现运行界面如图 11-34 所示。

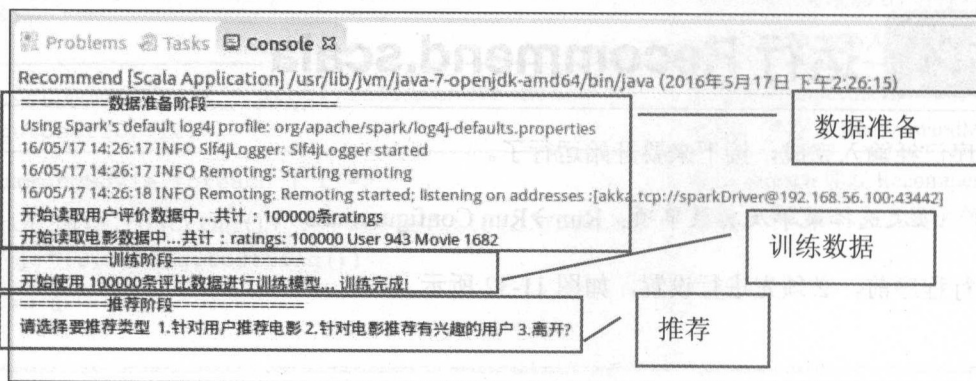


图 11-34 运行界面

如图 11-34 所示,运行后会显示数据准备阶段与训练阶段,最后进入推荐阶段,让用户输入选择。

步骤 04 针对用户推荐电影

如图 11-35 所示,选择“1.针对用户推荐电影”,然后输入用户 id 196。

程序就会显示推荐的产品,评分越高代表针对此用户越优先推荐此电影,评分从大到小排序;也就是说第 1 条数据,是系统推荐评分最高的电影。

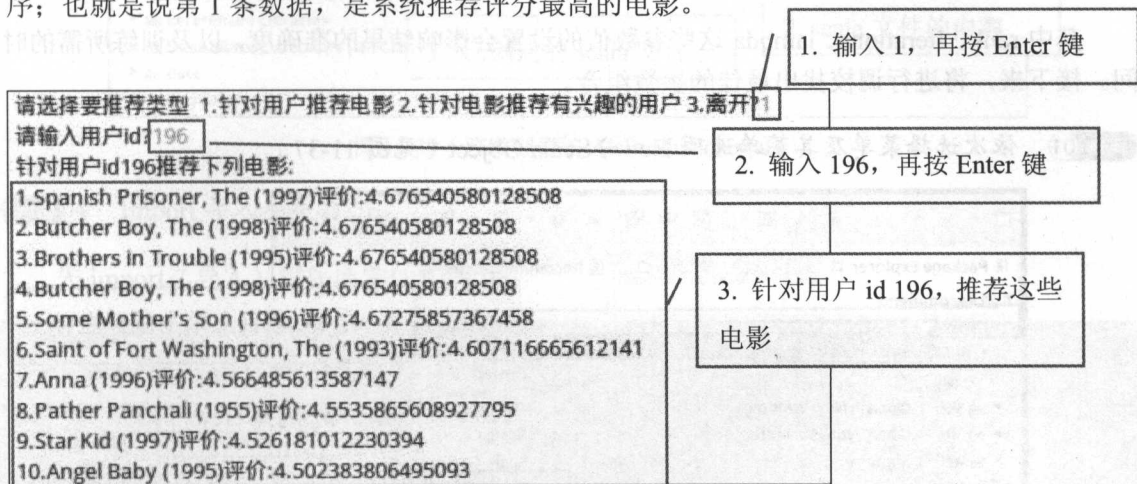


图 11-35 针对用户推荐电影

步骤 05 针对电影推荐给感兴趣的用户

如图 11-36 所示,如果选择“2.针对电影推荐有兴趣的用户”,然后输入电影的 id: 100,程序就会显示感兴趣的用户;评分越高,代表此用户越感兴趣。评分从大到小排序,也就是说第 1 条数据即为最感兴趣的用户。

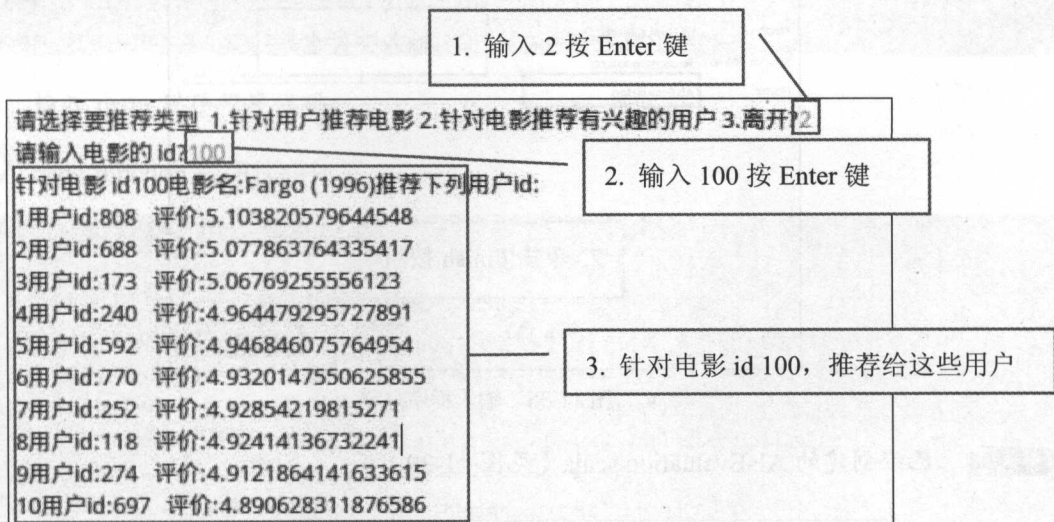


图 11-36 针对电影推荐给感兴趣的用户

11.15 创建 AlsEvaluation.scala 调校推荐引擎参数

在前面的章节中，我们使用 ALS.train 命令进行训练，会返回 model 训练完成的模型：

```
ALS.train(ratings: RDD[Rating], rank: Int, iterations: Int, lambda: Double):  
MatrixFactorizationModel
```

其中 rank、Iterations、lambda 这些参数值的设置会影响结果的准确度，以及训练所需的时间。接下来，将进行调校找出最佳的参数组合。

步骤 01 依次选择菜单及其菜单选项 New→Scala Object（见图 11-37）

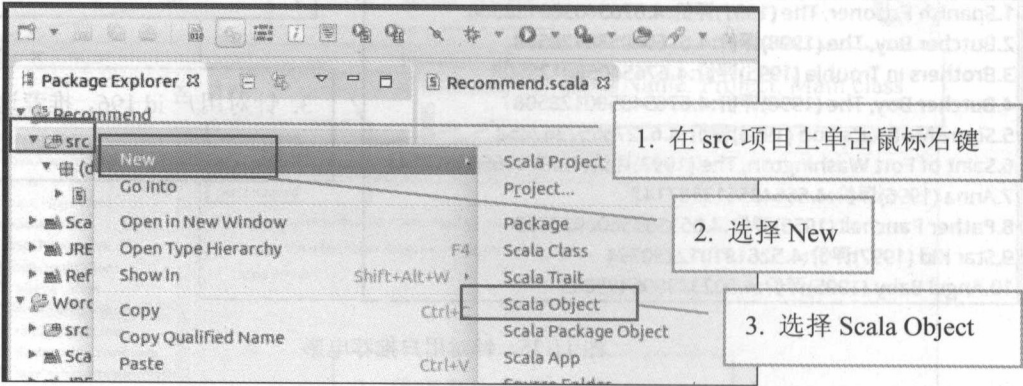


图 11-37 依次选择菜单及其菜单选项 New→Scala Object

步骤 02 输入程序名称（见图 11-38）

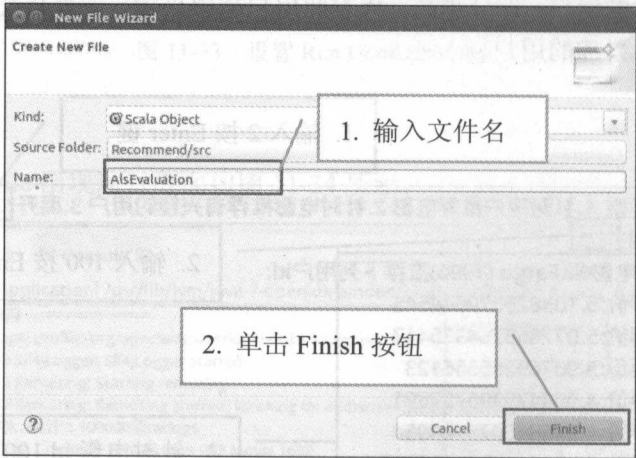


图 11-38 输入程序名称

步骤 03 已经创建的 AlsEvaluation.scala（见图 11-39）

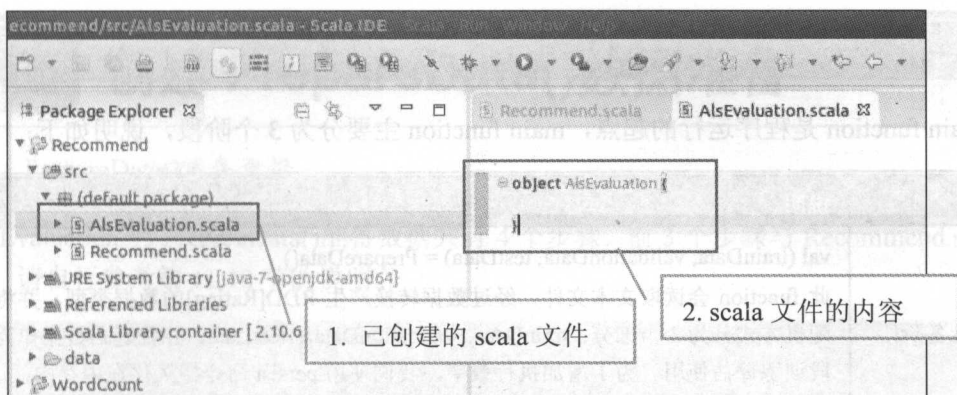


图 11-39 已经创建的 AlsEvaluation.scala

步骤 04 Import 导入相关的 lib

先 import (导入) 程序所需的相关 lib:

```
import java.io.File
import scala.io.Source
import org.apache.log4j.Logger
import org.apache.log4j.Level
import org.apache.spark.SparkConf
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.rdd._
import org.apache.spark.mllib.recommendation.{ALS, Rating,
MatrixFactorizationModel}
import org.joda.time.format._
import org.joda.time._
import org.joda.time.Duration
import org.jfree.data.category.DefaultCategoryDataset
import org.apache.spark.mllib.regression.LabeledPoint
```

步骤 05 输入 main 程序程序代码

输入 main function 程序代码如下:

```
def main(args: Array[String]) {
  SetLogger
  println("=====数据准备阶段=====")
  val (trainData, validationData, testData) = PrepareData()
  trainData.persist(); validationData.persist(); testData.persist()
  println("=====训练验证阶段=====")
  val bestModel = trainValidation(trainData, validationData)
  println("=====测试阶段=====")
  val testRmse = computeRMSE(bestModel, testData)
  println("使用 testData 测试 bestModel, " + "结果 rmse = " + testRmse)
```

```
trainData.unpersist(); validationData.unpersist(); testData.unpersist()
}
```

main function 是程序运行的起点，main function 主要分为 3 个阶段，说明如下：

阶段	说明
数据准备阶段	<code>val (trainData, validationData, testData) = PrepareData()</code> 此 function 会读取文本文件，经过数据转换产生 RDD[Rating]的数据类型，并将数据以随机方式分为 3 个部分：trainData、validationData、testData，并且返回数据作为下一阶段训练评估使用。为了增加执行效率，我们使用 persist 命令持久化在内存中： <code>trainData.persist(); validationData.persist(); testData.persist()</code>
训练评估阶段	<code>val model = trainEvaluate(trainData, validationData)</code> 此 function 传入 trainData（训练数据）、validationData（验证数据），程序会使用 trainData 进行训练产生 model 数据模型，并且使用 validationData 评估此数据模型的 RMSE（均方根误差）。参数的设置会影响误差，所以必须反复执行训练与评估，并找出最佳的参数组合。最后程序会返回训练完成的 bestModel 数据模型，作为下一阶段测试时使用
测试阶段	<code>testModel(model, testData)</code> 使用另外一组数据 testData 再测试一次，以避免出现 Overfitting 的问题（所谓 Overfitting 指的就是过度训练，意思就是说机器学习所学到的模型过度贴近 trainData，而导致和 testData 评估时误差变得更大。如果训练评估阶段时 RMSE 很低，但是测试阶段 RMSE 很高；代表可能有 Overfitting 的问题。如果测试与训练评估阶段，其结果 RMSE 差异不大，代表无 Overfitting 的问题）

以上 3 个阶段可以整理如图 11-40 所示。

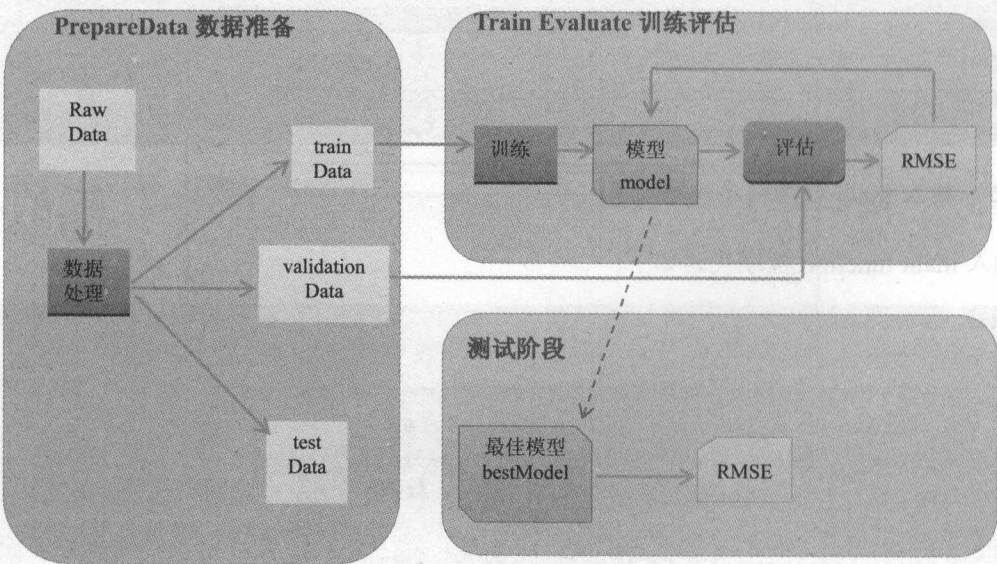


图 11-40 main function3 个阶段的执行流程图

11.16 创建 PrepareData()数据准备

步骤 01 PrepareData()准备数据

AlsEvaluation 的 PrepareData()准备数据共有 4 个步骤。前 3 个步骤与 Recommend.scala 完全相同，请自行参考第 11.12 节的说明。

```
def PrepareData(): (RDD[Rating], RDD[Rating], RDD[Rating]) = {  
  //-----1. 创建用户评分数据-----  
  //-----2. 创建电影 ID 与名称对照表-----  
  //-----3. 显示数据记录数-----  
  //-----4. 以随机方式将数据分为3个部分并且返回-----  
}
```

步骤 02 以随机方式将数据分为 3 个部分并且返回

第 4 步骤以随机方式将数据分为 3 个部分并且返回，详细说明如下：

```
//-----4. 以随机方式将数据分为3个部分并且返回-----  
println("将数据分为")  
val Array(trainData, validationData, testData) = ratingsRDD.randomSplit  
(Array(0.8, 0.1, 0.1))  
println(" trainData:" + trainData.count() + " validationData:"  
"+validationData.count() + " testData:" + testData.count())  
return (trainData, validationData, testData)  
}
```

将之前创建的 RDD[LabeledPoint]数据以随机方式，按照 8:1:1 比例随机方式分为 3 个部分：

- trainData（训练数据）
- validationData（验证数据）
- testData（测试数据）

上述程序代码的详细说明如下：

程序代码	说明
val Array(trainData, validationData, testData) = dataRDD.randomSplit(Array(0.8, 0.1, 0.1))	以 randomSplit 按照 8:1:1 比例随机方式分为 3 个部分： trainData、validationData、testData
println("将数据分 trainData:" + trainData.count() + " validationData:" + validationData.count() + " testData:" + testData.count())	输出 trainData、validationData、testData 的记录数
return (trainData, validationData, testData)	返回 trainData、validationData、testData 的数据

11.17 进行训练评估

步骤 01 trainValidation 训练评估

输入 trainValidation 函数程序代码如下：

```
def trainValidation(trainData: RDD[Rating], validationData: RDD[Rating]):  
MatrixFactorizationModel = {  
    println("-----评估 rank 参数使用 -----")  
    evaluateParameter(trainData, validationData, "rank", Array(5, 10, 15, 20,  
50, 100), Array(10), Array(0.1))  
    println("-----评估 numIterations -----")  
    evaluateParameter(trainData, validationData, "numIterations", Array(10),  
Array(5, 10, 15, 20, 25), Array(0.1))  
    println("-----评估 lambda -----")  
    evaluateParameter(trainData, validationData, "lambda", Array(10),  
Array(10), Array(0.05, 0.1, 1, 5, 10.0))  
    println("-----所有参数交叉评估找出最好的参数组合-----")  
    val bestModel = evaluateAllParameter(trainData, validationData, Array(5, 10,  
15, 20, 25), Array(5, 10, 15, 20, 25), Array(0.05, 0.1, 1, 5, 10.0))  
    return (bestModel)  
}
```

在 trainValidation 函数中，评估的参数共有 3 个 rank、numIterations、lambda。我们希望知道不同的参数值对 RMSE 的影响，以及执行所需要的时间。评估的方法是在同时间只评估一种参数。

➤ 评估 rank 参数

首先要评估 rank 参数时，固定 numIterations = Array(10)、lambda = Array(0.1)，但是 rank 参数有 6 个不同值 Array(5, 10, 15, 20, 50, 100)，用来评估哪一个 rank 参数值具有比较小的误差 RMSE。

程序代码	说明
<pre>evaluateParameter(trainData, validationData, "rank", Array(5, 10, 15, 20, 50, 100), Array(10), Array(0.1))</pre>	<p>evaluateParameter 函数传入下列参数：</p> <p>trainData、validationData；</p> <p>设置当前评估的目标是 rank 参数；</p> <p>rank 参数有 6 个不同值 Array(5, 10, 15, 20, 50, 100)；</p> <p>numIterations 参数固定是 10；</p> <p>lambda 参数固定是 0.1</p>

➤ 评估 numIterations 参数

当要评估 numIterations 参数时，固定 rank =Array(10)、lambda = Array(0.1)。但是 numIterations 参数有 5 个不同值 Array(5, 10, 15, 20, 25)，用来评估哪一个 numIterations 参数值具有比较小的误差 RMSE。

程序代码	说明
<pre>evaluateParameter(trainData, validationData, "numIterations", Array(10), Array(5, 10, 15, 20, 25), Array(0.1))</pre>	<p>evaluateParameter 函数，传入下列参数：</p> <p>trainData、validationData；</p> <p>设置当前评估的目标是 numIterations 参数；</p> <p>rank 参数固定是 10；</p> <p>numIterations 输入参数有 5 个不同值(5, 10, 15, 20, 25)；</p> <p>lambda 参数固定是 0.1</p>

➤ 评估 lambda 参数

当要评估 lambda 参数的方法时，固定 rank= Array(10)、numIterations= Array(10)。但是 lambda 输入参数有 5 个不同值(0.05, 0.1, 1, 5, 10.0)，用来评估哪一个 lambda 参数值具有比较小的误差 RMSE。

程序代码	说明
<pre>evaluateParameter(trainData, validationData, "lambda", Array(10), Array(10), Array(0.05, 0.1, 1, 5, 10.0))</pre>	<p>evaluateParameter 函数传入下列参数：</p> <p>trainData、validationData；</p> <p>设置当前评估的目标是 lambda 参数；</p> <p>rank 参数固定是 10；</p> <p>numIterations 参数固定是 10；</p> <p>lambda 输入参数有 5 个不同值(0.05, 0.1, 1, 5, 10.0)</p>

➤ 所有参数交叉评估

使用 evaluateAllParameter 函数进行所有参数交叉评估，找出最好的参数组合并传入下列参数。

程序代码	说明
<pre>val bestModel = evaluateAllParameter(trainData, validationData, Array(5, 10, 15, 20, 25), Array(5, 10, 15, 20, 25), Array(0.05, 0.1, 1, 5, 10.0))</pre>	<p>evaluateParameter 函数传入下列参数：</p> <p>trainData、validationData；</p> <p>rankArray 输入参数有 5 个不同值 Array(5, 10, 15, 20, 25)；</p> <p>numIterationsArray 输入参数有 5 个不同值 Array(5, 10, 15, 20, 25)；</p> <p>lambdaArray 输入参数有 5 个不同值 Array(0.05, 0.1, 1, 5, 10.0)；</p> <p>运行后将返回最佳的模型</p>

以上 3 个参数共有 $5 \times 5 \times 5 = 125$ 个排列组合。我们会对每一种参数组合评估 RMSE，最后找出具有最小误差 RMSE 的参数组合就是最佳组合。关于 evaluateAllParameter 函数在后续章节会详细说明。

步骤 02 evaluateParameter 评估单个参数

此函数主要是评估单个参数，哪一个参数值具有比较低的误差，并且绘制图形。

```
def evaluateParameter(trainData: RDD[Rating], validationData: RDD[Rating],
  evaluateParameter: String, rankArray: Array[Int], numIterationsArray:
  Array[Int], lambdaArray: Array[Double]) =
{
  var dataBarChart = new DefaultCategoryDataset()
  var dataLineChart = new DefaultCategoryDataset()
  for (rank <- rankArray; numIterations <- numIterationsArray; lambda <-
  lambdaArray) {
    val (rmse, time) = trainModel(trainData, validationData, rank,
    numIterations, lambda)
    val parameterData =
      evaluateParameter match {
        case "rank" => rank;
        case "numIterations" => numIterations;
        case "lambda" => lambda
      }
    dataBarChart.addValue(rmse, evaluateParameter,
    parameterData.toString())
    dataLineChart.addValue(time, "Time", parameterData.toString())
  }
  Chart.plotBarLineChart("ALS evaluations " + evaluateParameter,
  evaluateParameter, "RMSE", 0.58, 5, "Time", dataBarChart, dataLineChart)
}
```

上述程序代码的详细说明如下：

程序代码	说明
def evaluateParameter(trainData: RDD[Rating], validationData: RDD[Rating], evaluateParameter: String, rankArray: Array[Int], numIterationsArray: Array[Int], lambdaArray: Array[Double])	evaluateParameter 函数传入下列参数： trainData 训练数据； validationData 验证数据； 设置目前评估的参数，例如目前评估的参数是"rank" rank 参数数组； maxdepth 参数数组； maxBins 参数数组
var dataBarChart = new DefaultCategoryDataset()	创建柱形图所需要的数据集
var dataLineChart = new DefaultCategoryDataset()	创建折线图所需要数据集

(续表)

程序代码	说明
<pre>for (rank <- rankArray; numIterations <- numIterationsArray; lambda <- lambdaArray) {</pre>	<p>for 命令会重复执行 Array 中的每一个值:</p> <p>例如要评估 rank 参数的方法是固定 numIterations = Array(10)、lambda = Array(0.1), 但是 rank 参数有 6 个不同值 Array(5, 10, 15, 20, 50, 100), 所以程序会执行 6 次来评估哪一个参数值具有比较低的误差 RMSE</p>
<pre>val (rmse, time) = trainModel(trainData, validationData, rank, numIterations, lambda)</pre>	<p>执行 ALS 训练:</p> <p>传入参数 trainData (训练数据) 与 validationData (验证数据) rank、numIterations、lambda, 并返回此参数组合的 RMSE、训练所需时间</p>
<pre>val parameterData = evaluateParameter match { case "rank" => rank; case "numIterations" => numIterations; case "lambda" => lambda }</pre>	<p>获取当前评估的参数值:</p> <p>按照传入的 evaluateParameter 设置当前评估的参数, 获取当前的值; 例如当前评估的参数是 "rank", 就返回 rank 的值</p>
<pre>dataBarChart.addValue(rmse, evaluateParameter, parameterData.toString())</pre>	<p>加入柱形图数据, 柱形图用于显示 RMSE 显示误差</p>
<pre>dataLineChart.addValue(time, "Time", parameterData.toString())</pre>	<p>加入折线图数据, 折线图用于显示执行训练所需时间</p>
<pre>Chart.plotBarLineChart("ALS evaluations " + evaluateParameter, evaluateParameter, "RMSE", 0.58, 5, "Time", dataBarChart, dataLineChart)</pre>	<p>画出柱形图与折线图</p>

步骤 03 Chart.plotBarLineChart 绘制出柱形图与折线图

以下绘图程序主要是使用 jfreeChart 绘制柱形图与折线图, 参考程序代码注释:

```
import org.jfree.chart._
import org.jfree.data.xy._
```



```

import org.jfree.data.category.DefaultCategoryDataset
import org.jfree.chart.axis.NumberAxis
import org.jfree.chart.axis._
import Java.awt.Color
import org.jfree.chart.renderer.category.LineAndShapeRenderer;
import org.jfree.chart.plot.DatasetRenderingOrder;
import org.jfree.chart.labels.StandardCategoryToolTipGenerator;
import Java.awt.BasicStroke

object Chart {
  def plotBarLineChart(Title: String, xLabel: String, yBarLabel: String,
    yBarMin: Double, yBarMax: Double, yLineLabel: String, dataBarChart :
    DefaultCategoryDataset, dataLineChart: DefaultCategoryDataset): Unit = {

    //画出 Bar Chart
    val chart = ChartFactory
      .createBarChart(
        "", // Bar Chart 标题
        xLabel, // X 轴标题
        yBarLabel, // Bar Chart 标题 y 轴标题 1
        dataBarChart , // Bar Chart 数据
        org.jfree.chart.plot.PlotOrientation.VERTICAL, //画图方向垂直
        true, // 包含 legend
        true, // 显示 tooltips
        false // 不要 URL generator
      );
    //获得 plot
    val plot = chart.getCategoryPlot();
    plot.setBackgroundPaint(new Color(0xeee, 0xee, 0xeee));
    plot.setDomainAxisLocation(AxisLocation.BOTTOM_OR_RIGHT);
    plot.setDataset(1, dataLineChart); plot.mapDatasetToRangeAxis(1, 1)
    //画柱形图 y 轴
    val vn = plot.getRangeAxis(); vn.setRange(yBarMin, yBarMax);
    vn.setAutoTickUnitSelection(true)
    //画折线图 y 轴
    val axis2 = new NumberAxis(yLineLabel); plot.setRangeAxis(1, axis2);
    val renderer2 = new LineAndShapeRenderer()
    renderer2.setToolTipGenerator(new StandardCategoryToolTipGenerator());
    //设置先画柱形图,再画折线图以免折线图被盖掉
    plot.setRenderer(1,
    renderer2); plot.setDatasetRenderingOrder(DatasetRenderingOrder.FORWARD);
    //创建画框
    val frame = new ChartFrame(Title, chart); frame.setSize(500, 500);
    frame.pack(); frame.setVisible(true)
  }
}

```


步骤 04 trainModel 训练模型

输入下列 trainModel 训练模型程序代码。

```
def trainModel(trainData: RDD[Rating], validationData: RDD[Rating], rank: Int,
iterations: Int, lambda: Double): (Double, Double) = {
    val startTime = new DateTime()
    val model = ALS.train(trainData, rank, iterations, lambda)
    val endTime = new DateTime()
    val Rmse = computeRmse(model, validationData)
    val duration = new Duration(startTime, endTime)
    println(f" 训练参数: rank:$rank%3d,iterations:$iterations%.2f ,lambda =
$lambda%.2f 结果 Rmse=$Rmse%.2f" + "训练需要时间" + duration.getMillis + "毫秒")
    (Rmse, duration.getStandardSeconds)
}
```

上述程序代码 d 详细说明如下：

程序代码	说明
def trainModel(trainData: RDD[Rating], validationData: RDD[Rating], rank: Int, iterations: Int, lambda: Double): (Double, Double) = {	定义 trainModel function 传入下列参数： 训练数据、验证数据； rank 参数； iterations 参数； lambda 参数； 返回 RMSE 与训练所需时间
val startTime = new DateTime()	记录开始进行训练的时间
val model = ALS.train(trainData, rank, iterations, lambda)	使用 trainData 进行 ALS 训练，并且传入参数 rank、iterations、lambda
val endTime = new DateTime()	记录完成训练的时间
val duration = new Duration(startTime, endTime)	传入开始时间与结束时间，计算训练所需时间
println(f"训练参数: rank:\$rank%3d,iterations: \$iterations%.2f,lambda = \$lambda%.2f 结果 Rmse= \$Rmse%.2f" + "训练需要时间" + duration.getMillis + "毫 秒")	在字符串前面加上 "f" 字符串"，代表格式化的 f 字符串。 例如：“\$rank%3d”会输出 rank 常数（以 3 个位数显示），“\$iterations%.2f”会输出 iterations 常数（以小数点 2 位数显示）
(Rmse, duration.getStandardSeconds)	返回 RMSE 与训练所需时间： 在 scala 语言中可以使用 return 关键词返回，但是也可以省略 return 关键词；scala 语言会自动将函数的最后执行的一行语句当成是返回值

步骤 05 计算 RMSE

RMSE（Root Mean Square Error）等常用的统计工具，是用来计算推荐系统对用户喜好的预测，与用户实际喜好的误差平均值。通常 RMSE 越小代表误差越小，即代表预测值与真实

的值越接近、准确度越高。RMSE 公式如下：

$$RMSE = \sqrt{\frac{1}{N} \sum_{i=1}^N (y_i - x_i)^2}$$

其中 y_i : 预测值 x_i : 真实的值

我们创建下列函数，计算 RMSE（Root Mean Squared Error）：

```
def computeRMSE(model: MatrixFactorizationModel, RatingRDD: RDD[Rating]):
Double = {
    val num = RatingRDD.count()
    val predictedRDD = model.predict(RatingRDD.map(r => (r.user, r.product)))
    val predictedAndRatings =
        predictedRDD.map(p => ((p.user, p.product), p.rating))
        .join(RatingRDD.map(r => ((r.user, r.product), r.rating)))
        .values
    math.sqrt(predictedAndRatings.map(x => (x._1 - x._2) * (x._1 -
x._2)).reduce(_ + _) / num)
}
```

以上程序代码的详细说明如下：

程序代码	说明
def computeRMSE(model: MatrixFactorizationModel, RatingRDD: RDD[Rating]): Double = {	定义 computeRmse 传入下列参数： 使用测试数据训练完成的模型； 验证数据； 返回计算结果
val num = RatingRDD.count()	读取所有数据个数
val predictedRDD = model.predict(RatingRDD.map(r => (r.user, r.product)))	使用 model.predict 进行预测，将结果存入 predictedRDD； 传入参数 RatingRDD，使用.map 方法取出 user 与 product，并将结果存入 predictedRDD
val predictedAndRatings = predictedRDD.map(p => ((p.user, p.product), p.rating)) .join(RatingRDD.map(r => ((r.user, r.product), .rating))) .values	创建 predictedAndRatings： predictedRDD（预测结果）； join RatingRDD（真实数据）
math.sqrt(predictedAndRatings.map(x => (x._1 - x._2) * (x._1 - x._2)) .reduce(_ + _) / num)	计算 RMSE： predictedAndRatings 使用 map 针对每一条数据进行计算，并传入 x 作为参数。其中 x._1 是预测结果、x._2 是真实的数据，(x._1 - x._2)就是误差。误差相乘后就是平方，然后使用.reduce(_ + _)将全部误差平方求和后，再除以全部个数 num 以上计算结果，最后会以 math.sqrt 计算平方根

步骤 06 evaluateAllParameter 找出最佳的参数组合

evaluateAllParameter 程序将 3 个参数交叉评估找出最好的参数组合。输入 evaluateAllParameter 函数程序代码如下：

```
def evaluateAllParameter(trainData: RDD[Rating], validationData: RDD[Rating],
    rankArray: Array[Int], numIterationsArray: Array[Int], lambdaArray:
Array[Double]): MatrixFactorizationModel =
{
    val evaluations =
        for (rank <- rankArray; numIterations <- numIterationsArray; lambda <-
lambdaArray) yield {
            val (rmse, time) = trainModel(trainData, validationData, rank,
numIterations, lambda)
            (rank, numIterations, lambda, rmse)
        }
    val Eval = (evaluations.sortBy(_._4))
    val BestEval = Eval(0)
    println("最佳 model 参数: rank:" + BestEval._1 + ", iterations:" + BestEval._2
+ "lambda" + BestEval._3 + ",
    结果 rmse = " + BestEval._4)
    val bestModel = ALS.train(trainData, BestEval._1, BestEval._2,
BestEval._3)
    (bestModel)
}
```

上述函数在 trainValidation 函数中，我们的调用方式如下：

程序代码	说明
val bestModel = evaluateAllParameter(trainData, validationData, Array(5, 10, 15, 20, 25), Array(5, 10, 15, 20, 25), Array(0.05, 0.1, 1, 5, 10.0))	evaluateParameter 函数传入下列参数： trainData、validationData; rankArray 输入参数有 5 个不同值 Array(5, 10, 15, 20, 25); numIterationsArray 输入参数有 5 个不同值 Array(5, 10, 15, 20, 25); lambdaArray 输入参数有 5 个不同值 Array(0.05, 0.1, 1, 5, 10.0); 执行后返回最佳的模型

我们希望找出 3 种参数 rank、numIterations、lambda，交叉评估找出最好的参数组合。所以在 evaluateAllParameter 函数中，最重要的是 for 循环命令，程序代码的详细说明如下：

程序代码	说明
<pre>def evaluateAllParameter(trainData: RDD[Rating], validationData: RDD[Rating], rankArray: Array[Int], numIterationsArray: Array[Int], lambdaArray: Array[Double]) : MatrixFactorizationModel =</pre>	evaluateAllParameter 函数传入下列参数： trainData 训练数据； validationData 验证数据； rank 参数； numIterations 参数； lambda 参数； 返回 MatrixFactorizationModel 模型
<pre>val evaluations = for (rank <- rankArray; numIterations <- numIterationsArray; lambda <- lambdaArray) yield { ... }</pre>	for 循环有 3 个 Array : rankArray、numIterationsArray、lambdaArray; 3 个 Array 排列组合执行，共有 5*5*5=125 种排列组合
<pre>val (rmse, time) = trainModel(trainData, validationData, rank, numIterations, lambda)</pre>	执行 ALS 训练并返回训练完成的模型、训练所需时间。 传入参数: trainData(训练数据)、validationData(验证数据)、rank、numIterations、lambda
<pre>(rank, numIterations, lambda, rmse)</pre>	因为我们在 for 循环加入了 yield 关键词，每一次执行循环都会产生一组 (rank, numIterations, lambda, rmse)。
<pre>val Eval = (evaluations.sortBy(_._4)) val BestEval = Eval(0)</pre>	因为 RMSE 越低代表误差越小，所以我们希望在 125 条数据中找出 RMSE 最低的那笔数据。其中 “.sortBy(_._4)” 是 evaluationsArray 的第 4 个字段，也就是 RMSE 进行排序；默认从小到大排序。使用 “(0)” 读取排序后的第一笔数据，存入 BestEval 中
<pre>println(" 最佳 model 参数 : rank:" + BestEval._1 + ", iterations:" + BestEval._2 + "lambda" + BestEval._3 + ", 结果 rmse = " + BestEval._4)</pre>	屏幕显示最佳组合参数 rank、numIterations、lambda 与 RMSE
<pre>val bestModel = ALS.train(trainData, BestEval._1, BestEval._2, BestEval._3)</pre>	再次训练 ALS.train，参数传入：训练数据；最佳组合参数: rank、numIterations、lambda

11.18 运行 AlsEvaluation

步骤 01 依次选择菜单及其菜单选项 Run→Run Configurations

运行程序前必须先进行设置，如图 11-41 所示。

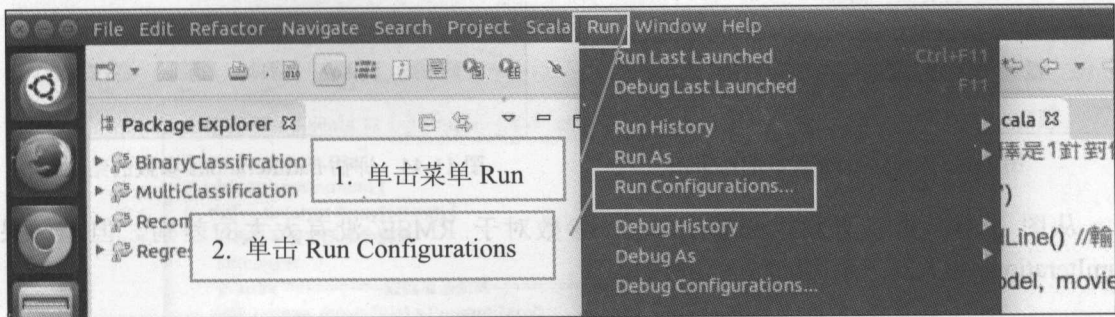


图 11-41 运行程序前必须先进行运行设置

步骤 02 运行 AlsEvaluation (见图 11-42)

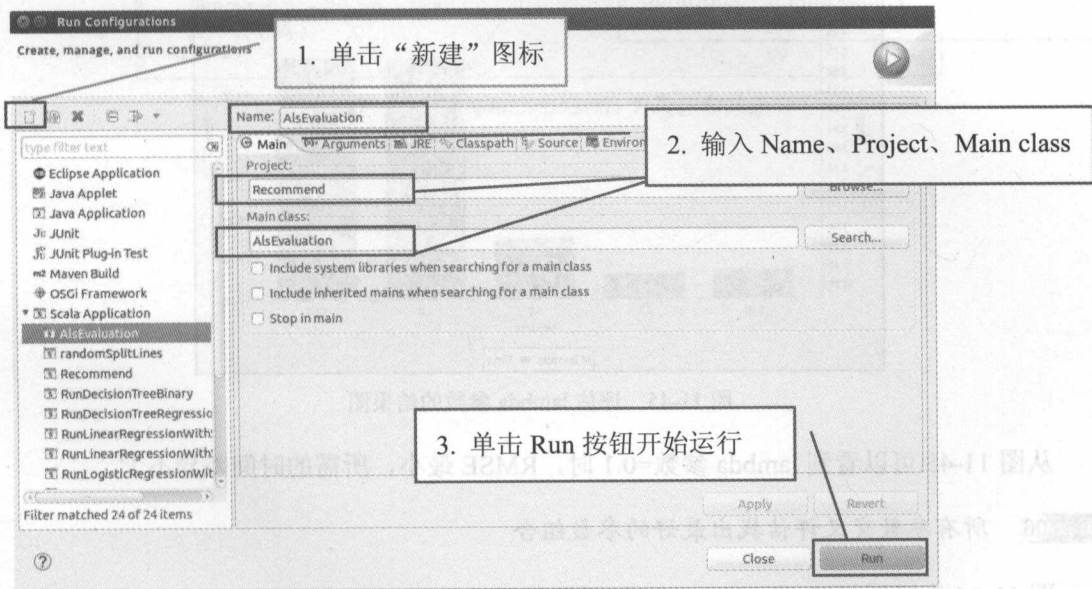


图 11-42 运行 AlsEvaluation

步骤 03 评估 rank 参数

运行后会显示如图 11-43 所示的图形。

柱形图代表 RMSE、折线图代表时间。从图 11-43 可以看到 rank 参数对于 RMSE 没有太大的差别。但是如果 rank 越大时，所需的时间会增加很多。

步骤 04 评估 numIterations 参数 (见图 11-44)

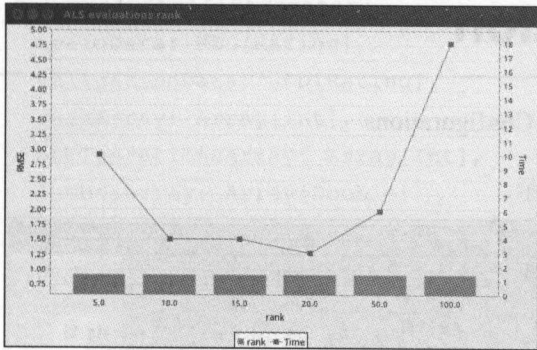


图 11-43 评估 rank 参数的结果图

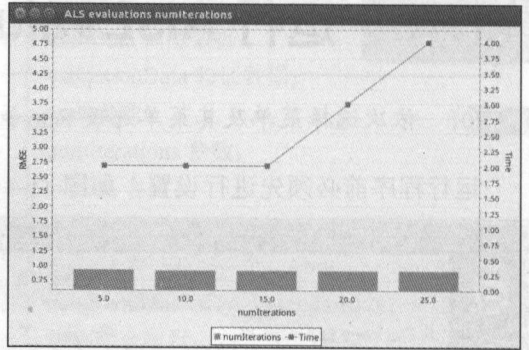


图 11-44 评估 numIterations 参数的结果图

从图 11-44 可以看到, numIterations 参数对于 RMSE 没有太大的差别。但是如果 numIterations 越大时, 所需的时间会增加很多。

步骤 05 评估 lambda 参数 (见图 11-45)

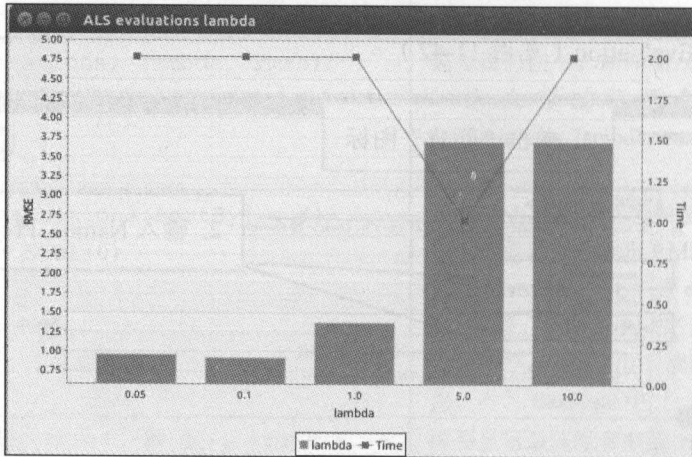


图 11-45 评估 lambda 参数的结果图

从图 11-45 可以看到 lambda 参数=0.1 时, RMSE 最小, 所需的时间差异不大。

步骤 06 所有参数交叉评估找出最好的参数组合

图 11-46 为运行 evaluateAllParameter 后屏幕显示界面。

```

最佳model参数: rank:20, iterations:15, lambda:0.1, 结果rmse = 0.9231243446391473
=====测试阶段=====
使用testData测试bestModel, 结果rmse = 0.9346648739589151
    
```

图 11-46 所有参数交叉评估找出最好的参数组合

最佳 model 的参数组合: rank:20 , iterations:15, lambda:0.1; 结果 rmse = 0.9231243446391473。最后使用 testData 再次验证 bestModel 结果 0.9346648739589151, 确认无 over fitting 问题。

11.19 修改 Recommend.scala 为最佳参数组合

既然我们已经找出最佳的参数组合，就可以修改 Recommend.scala 为最佳参数组合。修改下列训练参数为最佳参数组合，如图 11-47 所示。

```
val model = ALS.train(ratings, 20, 15, 0.1)
```

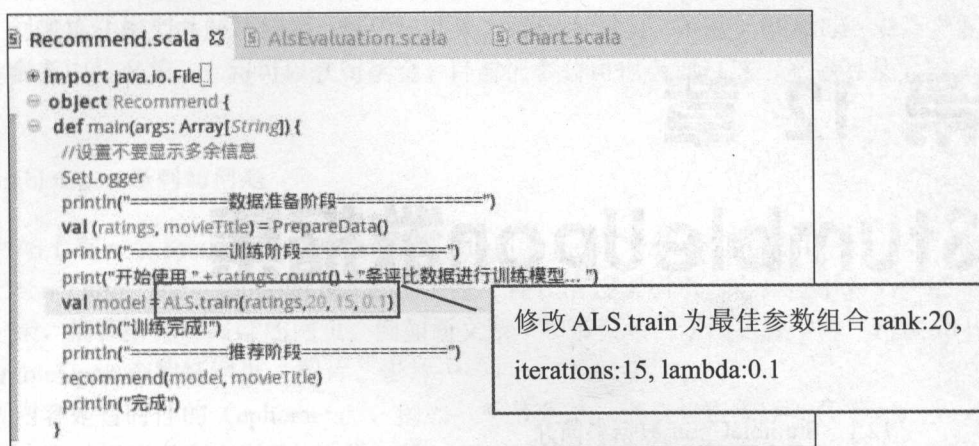


图 11-47 修改 Recommend.scala 为最佳参数组合

第 12 章

StumbleUpon数据集

- 12.1 StumbleUpon 数据集简介
- 12.2 下载 StumbleUpon 数据
- 12.3 用 LibreOffice Calc 电子表格查看 train.tsv
- 12.4 二元分类算法

12.1 StumbleUpon 数据集简介

步骤 01 Kaggle 网站介绍

Kaggle 是一个数据分析的竞赛平台, 网址为 <https://www.kaggle.com/>, 也是 Crowdsourcing (众包) 的平台。企业或研究者将大数据的问题发布到网站上, 包括数据、问题说明、期望的目标、奖金等, 来向大众征求解决方案。

网络上任何人都可以参与大数据问题的竞赛: 下载数据、分析数据、运用机器学习、数据挖掘等知识, 创建算法模型并解决问题, 最后将结果上传到网站上。如提交申请的结果符合要求, 并且在参赛者中排名第一, 将可以获得奖金。目前很多公司也会通过这个平台寻找大数据人才。

步骤 02 StumbleUpon 遇到的问题

Kaggle 网站上有一个 StumbleUpon Evergreen Classification Challenge 的题目。

StumbleUpon (<http://www.stumbleupon.com/>) 是个性化的搜索引擎, 它会按用户的兴趣和网页评分等记录, 推荐给你感兴趣的网页。例如新文章、季节菜单、新闻、教学等。超过数千万人使用 StumbleUpon 查找新网页、图片、影片……。

有些网页内容是暂时性的 (ephemeral), 例如: 季节菜单、当日股市涨跌新闻等。这些文章可能只是某一段时间读者会有兴趣, 过了这段时间对于读者就不太感兴趣。有些网页内容是长青的 (evergreen), 例如理财观念、育儿知识等, 这些文章读者长久都会感兴趣。

分辨网页是暂时性或是长青的, 对于 StumbleUpon 推荐网页给用户有很大的帮助。例如: 读者 A 买卖股票, 他可能会对于当日股市涨跌新闻感兴趣, 可是过了一周就对这则新闻没兴趣了。但是如果是理财观念的文章, 读者 A 可能长久都会有兴趣。

这些网页内容有人看过了, 就可以大致分类为暂时性或是长青。可是网页内容成千上万, 我们不可能有足够的人力去判断网页是暂时性或长青。不但成本太高, 也无法做到实时性。

此时机器学习 (Machine Learning) 就派得上用场了。我们的目标是利用机器学习, 通过大量网页数据进行训练来创建一个模型, 并使用这个模型去预测此网页是属于暂时性或长青的内容。

步骤 03 StumbleUpon 数据内容

你可以在下列网页查看 `stumbleupon` 数据的详细介绍:

<https://www.kaggle.com/c/stumbleupon/data>

在这里字段是以 0 为第 1 个字段。

➤ 字段 0~2

网址、网址 ID、样板文字 (见图 12-1), 这些字段与判断网页是暂时性 (ephemeral) 或

长青的关系不大，所以我们会忽略。

FieldName	Type	Description
url	string	Url of the webpage to be classified
urlid	integer	StumbleUpon's unique identifier for each url
boilerplate	json	Boilerplate text

图 12-1 字段 0~2

➤ 字段 3~25

Feature 特征字段：数值字段其内容是有关此网页的相关信息，例如：网页分类、链接的数目、图像的比例等，如图 12-2 和图 12-3 所示。

alchemy_category	string	Alchemy category (per the publicly available Alchemy API found at www.alchemyapi.com)
alchemy_category_score	double	Alchemy category score (per the publicly available Alchemy API found at www.alchemyapi.com)
avglinksiz	double	Average number of words in each link
commonLinkRatio_1	double	# of links sharing at least 1 word with 1 other links / # of links
commonLinkRatio_2	double	# of links sharing at least 1 word with 2 other links / # of links
commonLinkRatio_3	double	# of links sharing at least 1 word with 3 other links / # of links
commonLinkRatio_4	double	# of links sharing at least 1 word with 4 other links / # of links
compression_ratio	double	Compression achieved on this page via gzip (measure of redundancy)
embed_ratio	double	Count of number of <embed> usage
frameBased	integer (0 or 1)	A page is frame-based (1) if it has no body markup but have a frameset markup
frameTagRatio	double	Ratio of iframe markups over total number of markups
hasDomainLink	integer (0 or 1)	True (1) if it contains an <a> with an url with domain
html_ratio	double	Ratio of tags vs text in the page

图 12-2 字段 3~25 (一)

image_ratio	double	Ratio of tags vs text in the page
is_news	integer (0 or 1)	True (1) if StumbleUpon's news classifier determines that this webpage is news
lengthyLinkDomain	integer (0 or 1)	True (1) if at least 3 <a> 's text contains more than 30 alphanumeric characters
linkwordscore	double	Percentage of words on the page that are in hyperlink's text
news_front_page	integer (0 or 1)	True (1) if StumbleUpon's news classifier determines that this webpage is front-page news
non_markup_alphanum_characters	integer	Page's text's number of alphanumeric characters
numberOfLinks	integer	Number of <a> markups
numwords_in_url	double	Number of words in url
parametrizedLinkRatio	double	A link is parametrized if it's url contains parameters or has an attached onClick event
spelling_errors_ratio	double	Ratio of words not found in wiki (considered to be a spelling mistake)

图 12-3 字段 3~25 (二)

➤ 字段 26

这是 label，具有 2 个值（见图 12-4）。

- 1：代表长青（evergreen）——此网页会持续让用户感兴趣。
- 0：代表 non-evergreen——此网页暂时性。

label	integer (0 or 1)	User-determined label. Either evergreen (1) or non-evergreen (0), available for train.tsv only
-------	------------------	--

图 12-4 字段 26

12.2 下载 StumbleUpon 数据

步骤 01 到下载网址

在浏览器输入下列网址，进入 Kaggle 网站的 StumbleUpon 页面。

<https://www.kaggle.com/c/stumbleupon/data>

在网页中，可以看到有 2 个文件：train.tsv（训练数据）、test.tsv（测试数据），我们将下载这 2 个文件。先示范下载 train.tsv（训练数据），如图 12-5 所示。

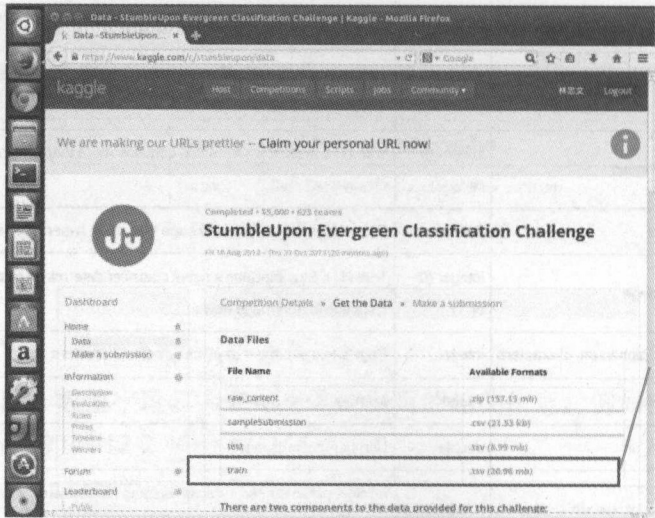


图 12-5 下载训练数据

步骤 02 注册网站

下载前必须先注册，可以使用邮箱账号进行注册，如图 12-6 所示。

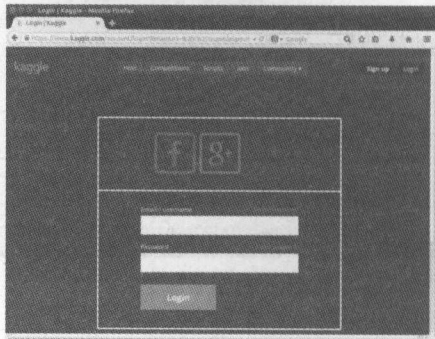


图 12-6 注册网站

步骤 03 下载 train.tsv (见图 12-7)

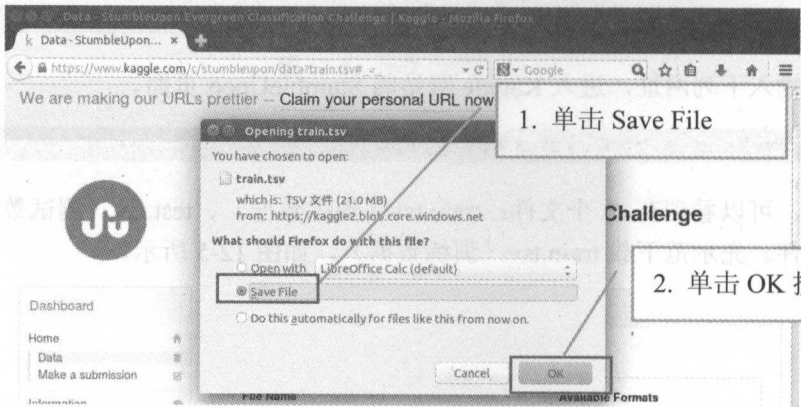


图 12-7 下载 train.tsv

步骤 04 打开所在的文件夹（见图 12-8）

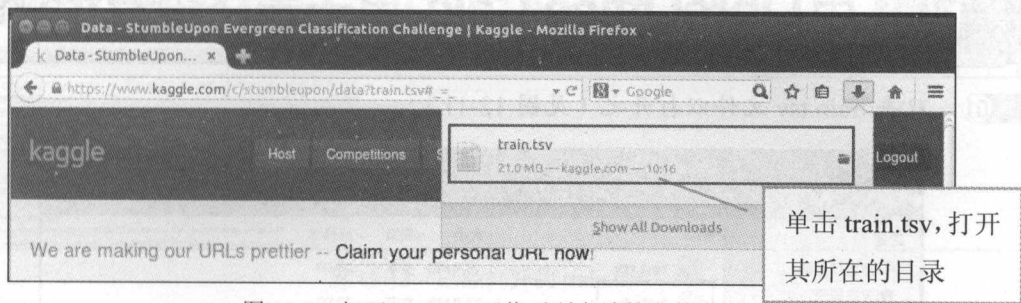


图 12-8 打开 train.tsv 下载后所存放的文件夹

步骤 05 下载 test.tsv

按照相同步骤下载 test.tsv 测试数据，如图 12-9 所示。

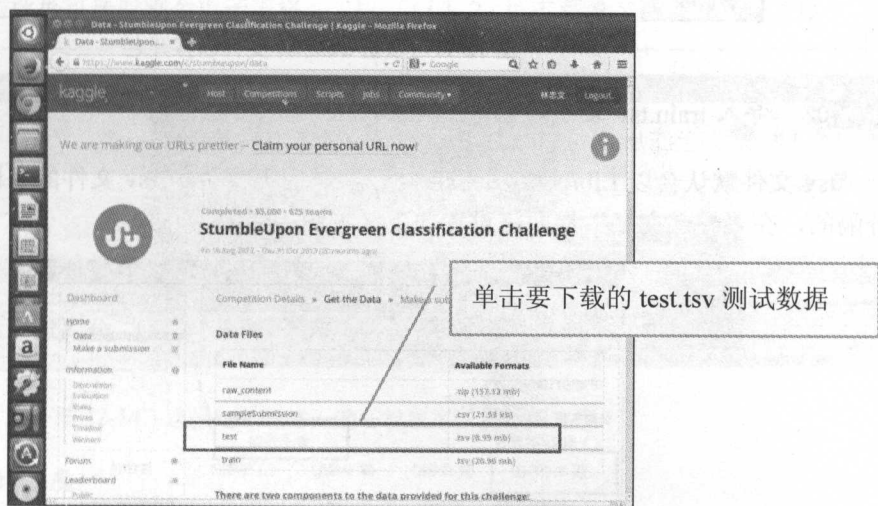


图 12-9 下载 test.tsv

步骤 06 查看已下载的文件

全部下载完成后，可以看到有 2 个文件：test.tsv（测试数据）、train.tsv（训练数据），如图 12-10 所示。

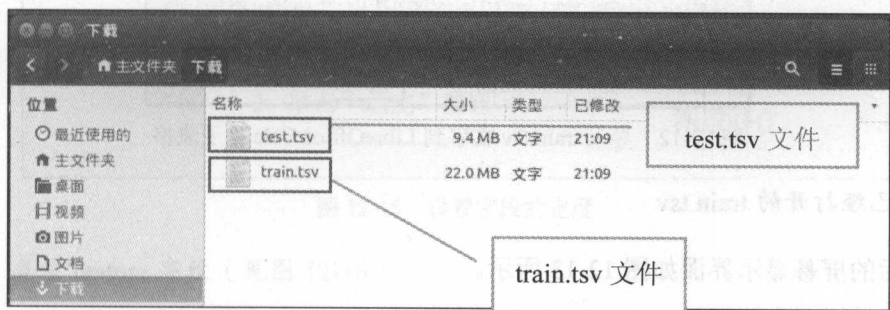


图 12-10 查看已下载的文件

12.3 用 LibreOffice Calc 电子表格查看 train.tsv

步骤 01 双击 train.tsv 文件以打开之（见图 12-11）

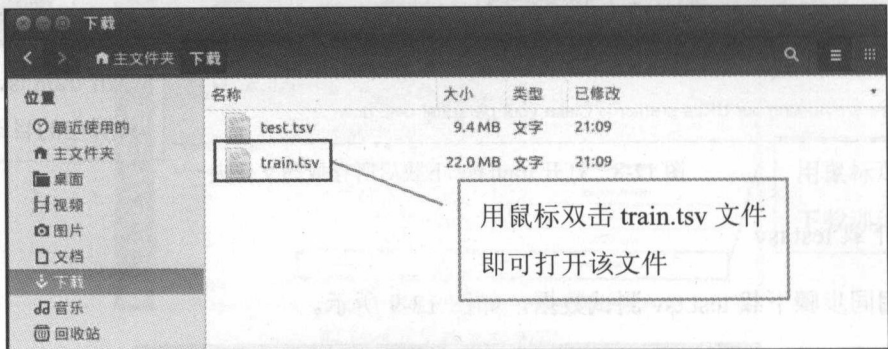


图 12-11 双击 train.tsv 文件以打开之

步骤 02 导入 train.tsv 文字到 LibreOffice Calc 电子表格

tsv 文件默认会以 LibreOffice Calc 电子表格打开文件。Tsv 文件的字段之间是以定制表符分隔的，在“导入文字”界面需设置如下（见图 12-12）。

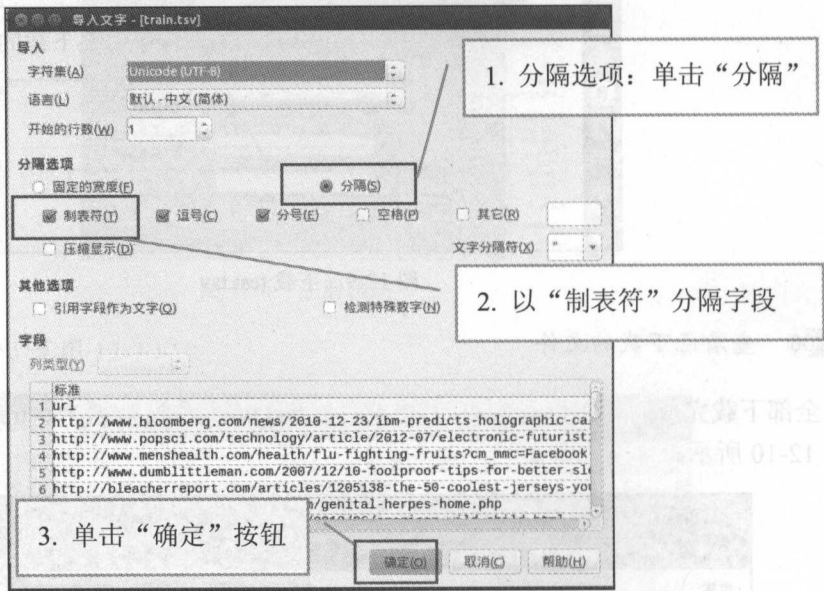


图 12-12 导入 train.tsv 文字到 LibreOffice Calc 电子表格

步骤 03 已经打开的 train.tsv

打开后的屏幕显示界面如图 12-13 所示。

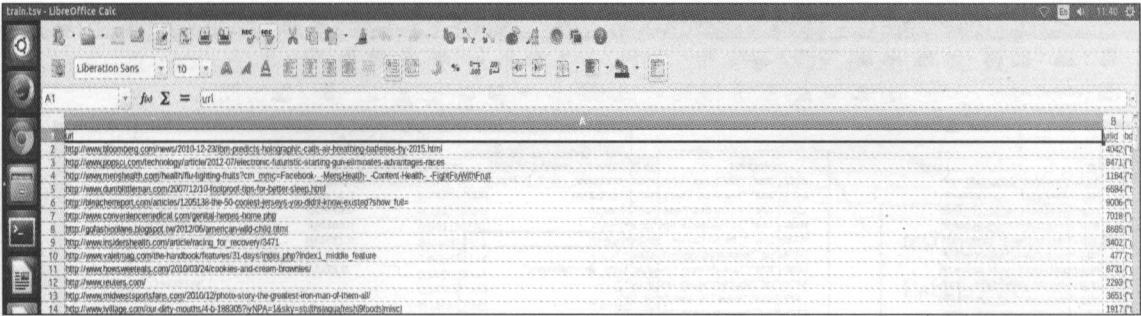


图 12-13 已经打开的 train.tsv

步骤 04 设置显示宽度

因为 url 字段太宽会影响阅读，所以设置所有字段固定宽度，如图 12-14 所示。

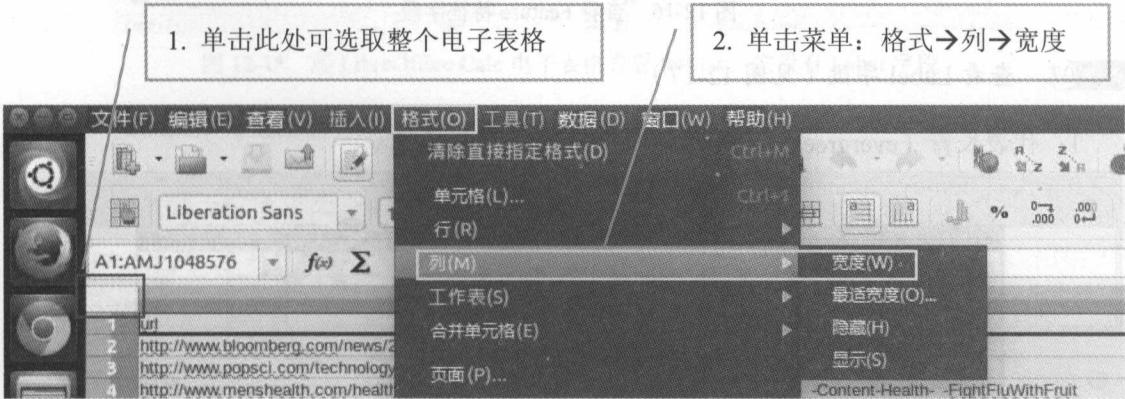


图 12-14 选择电子表格，准备设置字段显示的宽度

步骤 05 设置字段的宽度

设置所有字段固定宽度为 5 厘米，如图 12-15 所示。

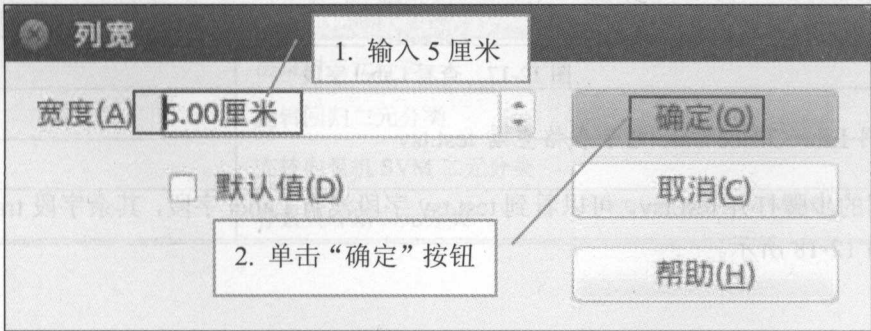


图 12-15 设置字段的宽度

步骤 06 查看 feature 字段（见图 12-16）

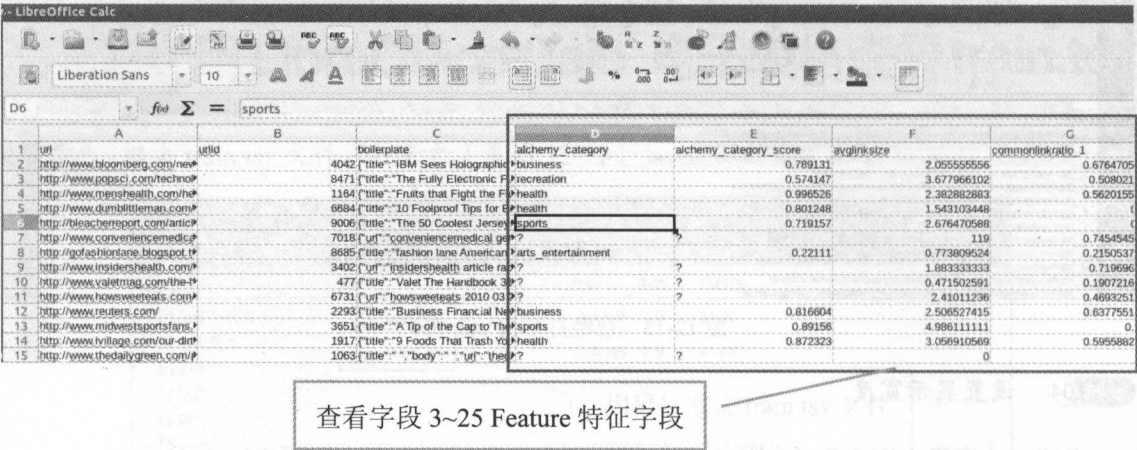


图 12-16 查看 Feature 特征字段

步骤 07 查看 Label 字段（见图 12-17）

- 1：代表长青（evergreen）——此网页会持续让用户感兴趣。
- 0：代表 non-evergreen——此网页暂时性。

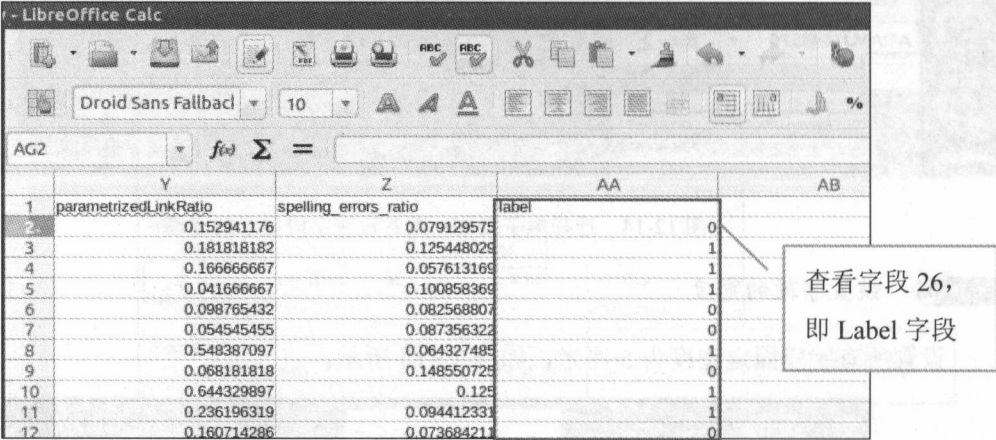


图 12-17 查看 Label 字段

步骤 08 用 LibreOffice Calc 电子表格查看 test.tsv

以相同的步骤打开 test.tsv。可以看到 test.tsv 字段没有 Label 字段，其余字段 train.tsv 完全相同，如图 12-18 所示。

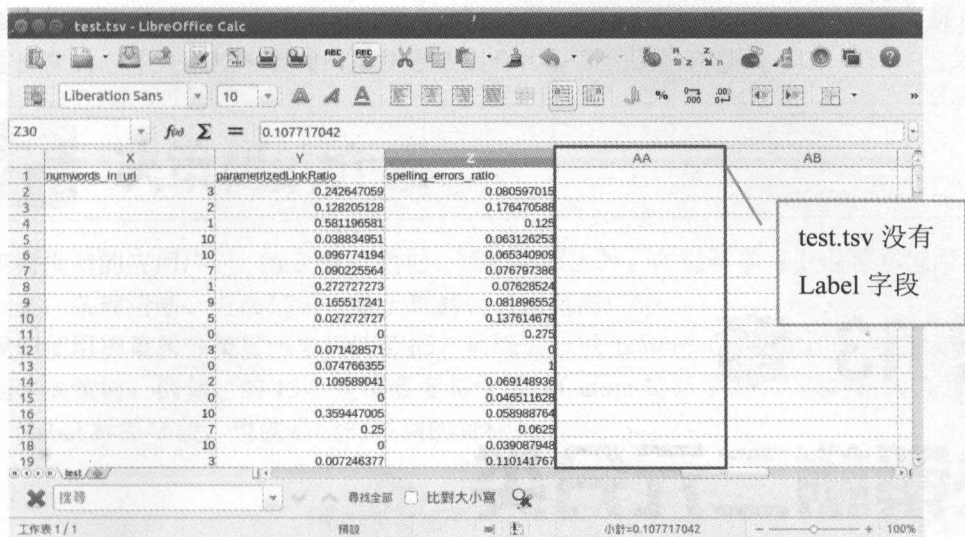


图 12-18 用 LibreOffice Calc 电子表格查看 test.tsv，它没有 Label 字段

步骤 09 train.tsv 与 test.tsv 比较

文件	说明
train.tsv	具有 feature（特征字段）与 label（标签字段），将用于训练数据
test.tsv	只有 feature（特征字段），将使用于预测数据

12.4 二元分类算法

接下来的章节，将示范如何使用下列二元分类算法来分析 StumbleUpon 数据集。预测哪些网页是暂时性，哪些是可以长久存在的，并且找出最佳参数组合，提高预测准确度。

章节	章节名称
13	决策树二元分类
14	逻辑回归二元分类
15	支持向量机 SVM 二元分类
16	朴素贝叶斯二元分类

第 13 章

决策树二元分类

- 13.1 决策树的介绍
- 13.2 创建 Classification 项目
- 13.3 开始输入 RunDecisionTreeBinary.scala 程序
- 13.4 数据准备阶段
- 13.5 训练评估阶段
- 13.6 预测阶段
- 13.7 运行 RunDecisionTreeBinary.scala
- 13.8 修改 RunDecisionTreeBinary 调校训练参数
- 13.9 运行 RunDecisionTreeBinary 进行参数调校
- 13.10 运行 RunDecisionTreeBinary 不进行参数调校

本章将使用第 12 章介绍的 StumbleUpon 数据集，运用决策树二元分类来预测网页是暂时性的或是长青的，并且调校参数找出最佳参数组合，提高预测准确度。

13.1 决策树的介绍

决策树模型的应用广泛，除了分析数据、预测模型之外，在机器学习上也有其应用。决策树的优点是：条理清晰、方法简单、易于理解、适用范围广等。

当我们使用决策树分类算法来训练数据，训练后会以 feature（特征字段）与 label（标签字段）创建决策树。例如，图 13-1 使用湿度与气压（feature 特征字段），来判断天气为“晴”或“雨”（label 标签字段，也就是我们预测的目标）。

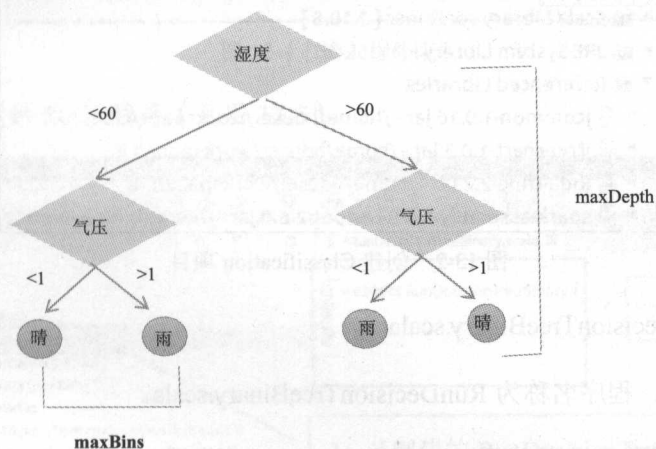


图 13-1 使用湿度与气压来判断天气为“晴”或“雨”的决策树

如图 13-1 所示，当我们使用历史数据执行训练时会创建决策树。可是决策树不可能无限成长，因此必须限制最大分支与深度。所以必须设置下列参数：

- **maxBins 参数**——决策树每一个节点的最大分支数目。
- **maxDepth 参数**——决策树的最大深度。
- **Impurity 参数**——决策树分裂节点时的方法。

当我们在父节点要分裂节点时，要以什么方法作为依据呢？例如湿度以 60 为分隔点，分为 >60 或 <60 ；或是湿度以 50 为分隔点，分为 >50 或 <50 。而到底哪一种方式比较好呢？此时有 Gini 与 Entropy 判断两种方式：

- **基尼系数 (Gini)**：由意大利统计学家 Corrado Gini 所发明，用于计算数值散布程度 (Statistical dispersion) 的指标。决策树算法对每种特征字段分隔点进行评估和计算，选择分裂后最小的基尼系数方式。
- **(Entropy) 熵**：熵被用于计算系统混乱的程度。决策树算法对每种特征字段分隔点进

行评估和计算，选择分裂后最小的熵方式。

13.2 创建 Classification 项目

步骤 01 创建 Classification 项目

参考第 11.4 节创建新的 scala 项目。项目名称为 Classification，如图 13-2 所示。

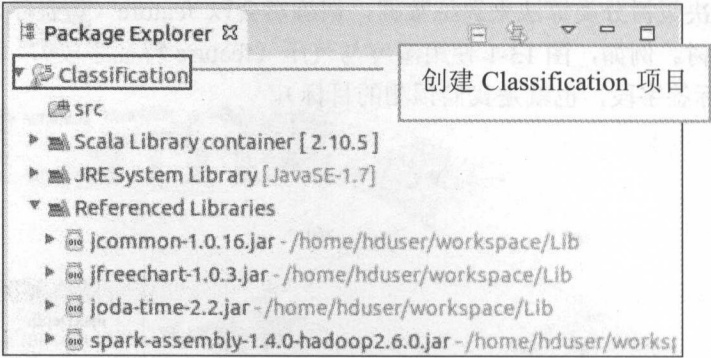


图 13-2 创建 Classification 项目

步骤 02 新建 RunDecisionTreeBinary.scala

创建 scala 程序，程序名称为 RunDecisionTreeBinary.scala。

步骤 03 依次选择菜单及其菜单项 New→Scala Object（见图 13-3）

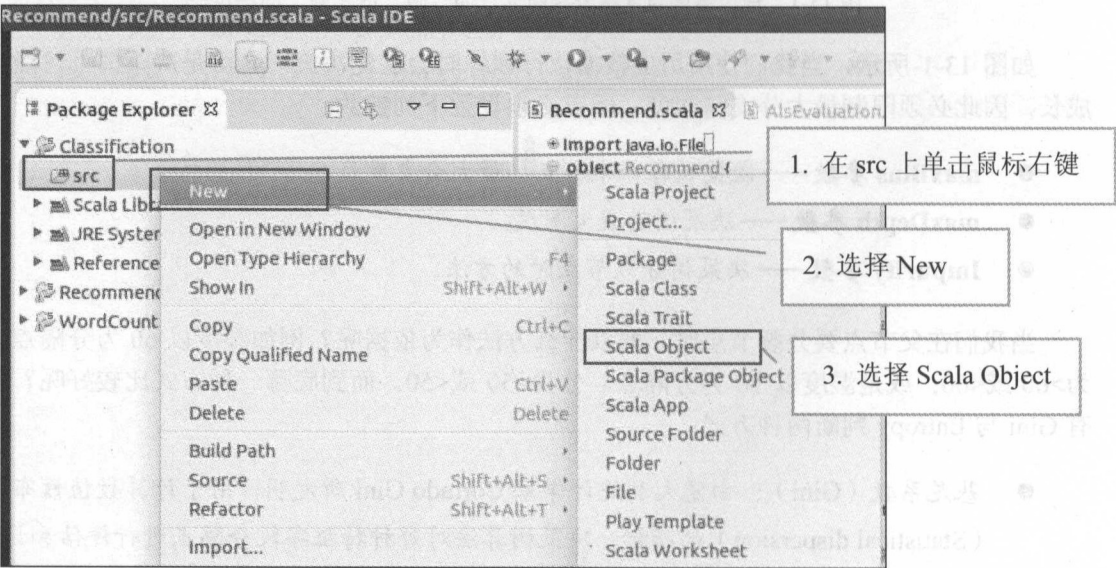


图 13-3 新建 RunDecisionTreeBinary.scala

步骤 04 输入程序名称（见图 13-4）

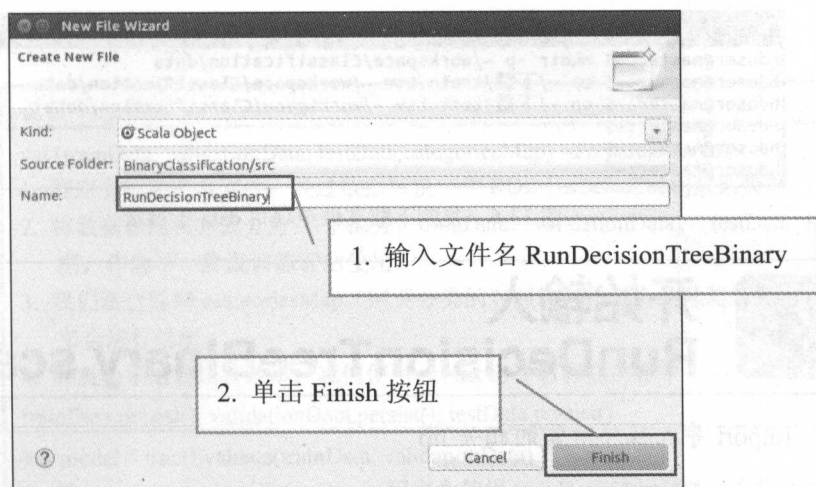


图 13-4 输入程序名称

步骤 05 已经创建的 Scala 程序（见图 13-5）

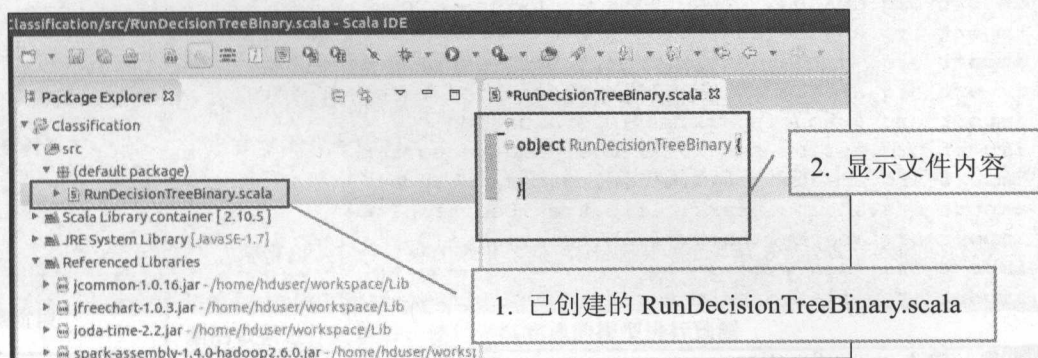


图 13-5 已创建的 RunDecisionTreeBinary.scala 程序

步骤 06 复制下载文件到项目 data 子目录

在“终端”程序中输入下列命令，将我们之前下载的 StumbleUpon 数据集文件复制到项目的 data 子目录。

➤ 创建项目的 data 子目录

```
mkdir -p ~/workspace/Classification/data
```

➤ 复制 train.tsv 到项目的 data 子目录

```
cp ~/下载/train.tsv ~/workspace/Classification/data
```

➤ 复制 test.tsv 到项目的 data 子目录

```
cp ~/下载/test.tsv ~/workspace/Classification/data
```

运行后屏幕显示界面如图 13-6 所示。

```

hduser@master:~$
hduser@master:~$ mkdir -p ~/workspace/Classification/data
hduser@master:~$ cp ~/下载/train.tsv ~/workspace/Classification/data
hduser@master:~$ cp ~/下载/test.tsv ~/workspace/Classification/data
hduser@master:~$
hduser@master:~$
hduser@master:~$

```

图 13-6 复制下载文件至项目 data 子目录

13.3

开始输入 RunDecisionTreeBinary.scala 程序

步骤 01 Import 导入程序所需的相关 lib

先 import 程序所需的相关 lib。

```

import org.apache.log4j.Logger
import org.apache.log4j.Level
import org.apache.spark.storage.StorageLevel
import org.apache.spark.rdd.RDD
import org.apache.spark.{ SparkConf, SparkContext }
import org.apache.spark.mllib.regression.LabeledPoint
import org.apache.spark.mllib.evaluation._
import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.mllib.feature.StandardScaler
import org.apache.spark.mllib.tree.DecisionTree
import org.apache.spark.mllib.tree.model.DecisionTreeModel
import org.joda.time._
import org.jfree.data.category.DefaultCategoryDataset

```

步骤 02 输入 main function

输入 main function 程序如下：

```

object RunDecisionTreeBinary {
  def main(args: Array[String]): Unit = {
    SetLogger()
    val sc = new SparkContext(new SparkConf().
setAppName("DecisionTreeBinary").setMaster("local[4]"))
    println("RunDecisionTreeBinary")
    println("=====数据准备阶段=====")
    val (trainData, validationData, testData, categoriesMap) = PrepareData(sc)
    trainData.persist(); validationData.persist(); testData.persist()
    println("=====训练评估阶段=====")
    val model = trainEvaluate(trainData, validationData)
    println("=====测试阶段=====")
    val auc = evaluateModel(model, testData)
    println("使用 testata 测试最佳模型, 结果 AUC:" + auc)
    println("=====预测数据=====")
    PredictData(sc, model, categoriesMap)
    trainData.unpersist(); validationData.unpersist(); testData.unpersist()
  }
}

```

main function 是程序运行的起点。main function 主要分为 4 个步骤，说明如下：

步骤	说明
数据准备阶段	<pre>val (trainData, validationData, testData, categoriesMap) = PrepareData(sc)</pre> <ol style="list-style-type: none">1. 程序会读取文本文件，经过数据转换产生 RDD [LabeledPoint]数据类型。2. 将数据以随机方式分为 3 个部分：trainData、validationData、testData 并返回数据，作为下一阶段训练评估使用。3. 我们还会返回 categoriesMap（网页分类的对照表）作为后续预测时使用，后续章节会进行说明。4. 并且为了增加运行效率，我们使用 persist 命令暂存在内存中 <code>trainData.persist(); validationData.persist(); testData.persist()</code>
训练评估阶段	<pre>var model = trainEvaluate(trainData, validationData)</pre> <ol style="list-style-type: none">1. 传入 trainData、validationData，程序会使用 trainData 进行训练，产生 model 数据模型。2. 使用 validationData 评估此数据模型准确度 AUC。3. 最后程序返回训练完成的 model 数据模型，作为下一阶段测试时使用
测试阶段	<pre>val auc = evaluateModel(model, testData)</pre> <ol style="list-style-type: none">1. 使用另外一组数据 testData 再次测试，以避免 overfitting 的问题。2. Overfitting（过度训练）。意思是说机器学习所学到的模型过度贴近 trainData，而导致误差变得更大。3. 如果训练评估阶段时 AUC 很高，但是测试阶段 AUC 很低，代表可能有 overfitting 的问题。4. 如果测试与训练评估阶段其结果 AUC 差异不大，代表无 overfitting 的问题
预测阶段	<pre>PredictData(sc, model, categoriesMap)</pre> <p>新的数据进行处理后，使用训练完成的模型进行预测</p>

我们整理 RunDecisionTreeBinary.scala 程序架构如图 13-7 所示。

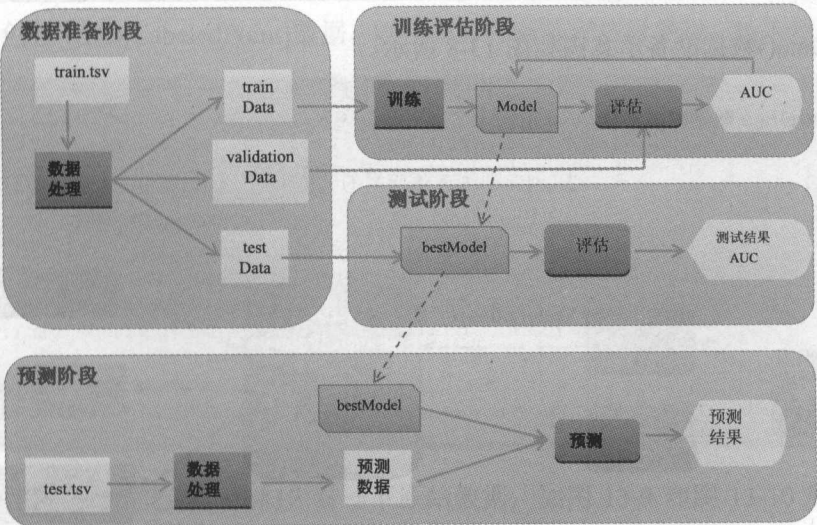


图 13-7 RunDecisionTreeBinary.scala 的程序架构图

13.4 数据准备阶段

步骤 01 创建 PrepareData():

数据准备主要分为 3 个步骤，先输入注释行：

```
def PrepareData(sc: SparkContext): (RDD[LabeledPoint], RDD[LabeledPoint],  
RDD[LabeledPoint], Map[String, Int]) = {  
  //-----1.导入/转换数据-----  
  //-----2.创建训练评估所需的数据 RDD[LabeledPoint]-----  
  //-----3.以随机方式将数据分为3个部分并且返回-----  
}
```

PrepareData()数据准备步骤说明如下：

数据准备步骤	说明
1.导入/转换数据	读取 train.tsv 并且去除第 1 行表头数据，并且以"t"分隔读取字段
2.创建 RDD[LabeledPoint]	后续我们进行训练与预测，都需要 LabeledPoint 数据类型的 RDD： LabeledPoint 是由 label 与 feature 组成，定义如下： LabeledPoint(label: Double, features: Array[Double]) 我们将在数据中提取 label 与 feature 字段，创建 RDD[LabeledPoint]
3.以随机方式将数据分为 3 个部分并且返回	将数据分为 3 个部分： ● trainData：作为训练数据产生模型 ● validationData：作为评估模型使用 ● testData：作为测试最后完成的模型

PrepareData()数据准备示意图如图 13-8 所示。

PrepareData 数据准备步骤

1. 导入/转换数据
2. 创建训练评估所需的数据
RDD[LabeledPoint]
3. 以随机方式将数据分为 3
个部分并且返回

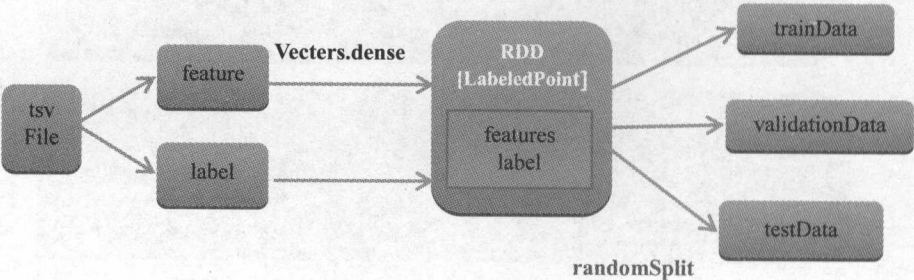


图 13-8 PrepareData()数据准备示意图

步骤02 导入转换数据

```
//-----1.导入并转换数据-----
print("开始导入数据...")
val rawDataWithHeader = sc.textFile("data/train.tsv")
val rawData = rawDataWithHeader.mapPartitionsWithIndex
  { (idx, iter) => if (idx == 0) iter.drop(1) else iter }
val lines = rawData.map(_.split("\t"))
println("共计: " + lines.count.toString() + "条")
```

上述命令的详细说明如下:

➤ 导入 train.tsv

```
val rawDataWithHeader = sc.textFile("data/train.tsv")
```

读取 train.tsv 并载入 rawDataWithHeader。

➤ 删除第一行表头

```
val rawData = rawDataWithHeader.mapPartitionsWithIndex
  { (idx, iter) => if (idx == 0) iter.drop(1) else iter }
```

因为 tsv 文件第一行数据是字段名不是数据, 所以我们必须删除第一行的表头。

➤ 读取每一行的数据字段

```
val lines = rawData.map(_.split("\t"))
```

因为文本文件是以 tab (制表符) 分隔字段, 所以使用 "\t" 分隔字段来读取每一个字段。

步骤03 创建训练评估所需数据

接下来要创建 RDD[LabeledPoint] 数据, 以作为后续训练评估时使用。输入程序代码如下:

```
//-----2.创建训练评估所需的数据 RDD[LabeledPoint]-----
val categoriesMap = lines.map(fields =>
  fields(3)).distinct.collect.zipWithIndex.toMap
val labelpointRDD = lines.map { fields =>
  val trFields = fields.map(_.replaceAll("\\", ""))
  val categoryFeaturesArray = Array.ofDim[Double](categoriesMap.size)
  val categoryIdx = categoriesMap(fields(3))
  categoryFeaturesArray(categoryIdx) = 1
  val numericalFeatures = trFields.slice(4, fields.size - 1)
    .map(d => if (d == "?") 0.0 else d.toDouble)
  val label = trFields(fields.size - 1).toInt
  LabeledPoint(label,
    Vectors.dense(categoryFeaturesArray ++ numericalFeatures))
}
```

在解释上述程序代码之前, 我们先看一下原始数据, 如图 13-9 和图 13-10 所示, 思考如何处理这些字段。

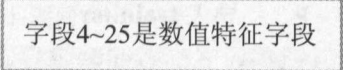


图 13-9 原始数据(一)

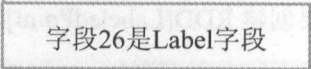


图 13-10 原始数据(二)

300

➤ 处理每一行数据:

```
val labelpointRDD = lines.map { fields =>
  ...
}
```

使用 `map` 针对每一条数据进行处理, 传入 `fields` 作为参数。数据处理后, 存储为 `labelpointRDD`。

➤ 删除双引号 “ ”:

```
val trFields = fields.map(_.replaceAll("\"", ""))
```

删除每一条数据中所有的双引号 “ ”。

➤ 字段 4~25 是 `numericalFeatures` 数值特征字段:

```
val numericalFeatures = trFields.slice(4, fields.size - 1).
map(d => if (d == "?") 0.0 else d.toDouble)
```

数值字段处理很简单, 我们会使用 `slice(4, fields.size - 1)` 来提取第 4 至倒数第 2 字段作为 `features`。如果数值是 “?” 则转换为 0。

➤ 字段 3 “分类特征字段” 转换为 “数值字段” 的方式:

字段 3 “分类特征字段” 转换为 “数值字段” 后, 产生 `categoryFeaturesArray` 这一部分比较复杂, 在下一步骤 (即步骤 4) 会详细介绍。

➤ label 字段

```
val label = trFields(fields.size - 1).toInt
```

提取最后一个字段作为 `label` 字段。

步骤 04 字段 3 “分类特征字段” 转换为 “数值字段” 的方式

因为字段 3 网页分类是分类特征 (Category Feature) 字段, 分类算法无法直接使用, 必须转换为数值字段才能够被分类算法使用。

转换的方法是 1-of-k encoding 的方式。如果网页分类有 N 个分类, 就会转换为 N 个数值字段。转换的方法如下列表格。为了方便说明, 我们假设网页分类只有 4 个分类, 因此 1 个分类特征字段会转换为 4 个数值字段。

如下列表格: 第 1 条数据分类特征的字段值是 `business`, 会转换为 4 个数值字段 1,0,0,0。

分类特征字段	数值字段 1	数值字段 2	数值字段 3	数值字段 4
Business	1	0	0	0
Recreation	0	1	0	0
Health	0	0	1	0
Sports	0	0	0	1

在程序中“分类特征字段”转换为“数值字段”的步骤如下：

➤ 创建网页分类对照表

```
val categoriesMap = lines.map(r => r(3)).distinct.collect.zipWithIndex.toMap
```

创建 categoriesMap: 先使用 lines.map(r => r(3)) 读取第 3 个字段、.distinct 删除重复、.collect 转换为 array、.zipWithIndex 转换为数字、.toMap 转换为对照表。

转换后结果如下: (business->0)、(recreation->1)、(health->2)、(sports->3)。因为我们假设网页分类只有 4 个分类, 所以对照表个数 categoriesMap.size 是 4。

➤ 定义 categoryFeaturesArray 数组

```
val categoryFeaturesArray = Array.ofDim[Double](categoriesMap.size)
```

我们使用上述命令定义 categoryFeaturesArray 数组, 数组的大小是 4(categoriesMap.size)。当我们创建一个新 Array 时, 其中的值默认都是 0。

➤ 网页分类转换为数值

```
val categoryIdx = categoriesMap(fields(3))
```

首先使用 categoriesMap 对照表传入字段 3 网页分类, 并且转换为数值。例如: 此数据分类是 health, 我们会转换为 categoryIdx=2。

➤ 设置 Array 相对应的位置是 1

```
categoryFeaturesArray(categoryIdx) = 1
```

然后设置 categoryFeaturesArray 数组相对应的位置是 1, 其余位置默认是 0。数组的位置是从 0 开始。

- 例如: 若字段 3 的值是 business, 经过对照表转换后 categoryIdx=0。所以我们会设置 categoryFeaturesArray 数组中第 0 个位置是 1, 结果是 1,0,0,0。
- 例如: 若字段 3 的值是 recreation, 经过对照表转换后 categoryIdx=1。所以我们会设置 categoryFeaturesArray 数组中第 1 个位置是 1, 结果是 0,1,0,0。
- 例如: 若字段 3 的值是 health, 经过对照表转换后 categoryIdx=2。所以我们会设置 categoryFeaturesArray 数组中第 1 个位置是 1, 结果是 0,0,1,0。

➤ 创建 LabeledPoint 并且返回

```
LabeledPoint(label, Vectors.dense(categoryFeaturesArray ++ numericalFeatures))
```

1. LabeledPoint 是由 label 与 feature 组成。
2. 提取最后一个字段作为 label 字段。
3. Feature 是由 categoryFeaturesArray (分类特征字段) ++ numericalFeatures (数值特征字

段) 数组相加而成。

步骤 05 以随机分割数据

接下来, 将之前创建的 RDD[LabeledPoint] 数据按随机方式分割为 3 个部分:

- trainData: 作为训练数据。
- validationData: 作为评估模型使用。
- testData: 作为测试预测数据使用。

```
//-----3.以随机方式将数据分为3部分并且返回-----
val Array(trainData, validationData, testData) =
  labelpointRDD.randomSplit(Array(0.8, 0.1, 0.1))
println("将数据分 trainData:" + trainData.count() +
  " validationData:" + validationData.count() +
  " testData:" + testData.count())
return (trainData, validationData, testData, categoriesMap) //返回数据
```

上述命令的详细说明如下:

➤ 以 randomSplit 随机方式分为 3 部分

```
val Array(trainData, validationData, testData) =
  labelpointRDD.randomSplit(Array(0.8, 0.1, 0.1))
```

以 randomSplit 按照 8:1:1 比例, 随机方式分为 3 部分: trainData、validationData、testData。

➤ 输出数据个数

```
println(
  "将数据分 trainData:" + trainData.count()
  + " validationData:" + validationData.count()
  + " testData:" + testData.count())
```

➤ 返回数据

```
return (trainData, validationData, testData, categoriesMap)
```

返回 trainData、validationData、testData、categoriesMap 数据。

13.5 训练评估阶段

步骤 01 trainEvaluate 训练评估

输入下列程序代码:

```
def trainEvaluate(trainData: RDD[LabeledPoint],
```

```
validationData: RDD[LabeledPoint]): DecisionTreeModel = {
  print("开始训练...")
  val (model, time) = trainModel(trainData, "entropy", 10, 10)
  println("训练完成,所需时间:" + time + "毫秒")
  val AUC = evaluateModel(model, validationData)
  println("评估结果 AUC=" + AUC)
  return (model)
}
```

trainEvaluate function 主要分为两个部分：训练模型、评估模型。

上述命令的详细说明如下：

➤ 定义 trainEvaluate

```
def trainEvaluate( trainData: RDD[LabeledPoint],
  validationData: RDD[LabeledPoint]): DecisionTreeModel = {
```

定义 trainEvaluate, 参数为：训练数据、验证数据、返回训练完成的模型。

➤ 训练模型

```
val (model, time) = trainModel(trainData, "entropy", 10, 10)
```

传入：trainData 训练数据与参数；返回：训练完成的数据模型、所需时间。

➤ 显示训练所需时间

```
println("训练完成,所需时间:" + time + "毫秒")
```

➤ 评估模型

```
val AUC = evaluateModel(model, validationData)
```

评估训练完成的数据模型 model 的准确率。对于二元分类，通常我们使用 AUC 进行评估。

➤ 返回训练完成的数据模型 model

```
return (model)
```

步骤 02 训练 DecisionTree 模型 Model

输入下列程序代码：

```
def trainModel(trainData: RDD[LabeledPoint], impurity: String,
  maxDepth: Int, maxBins: Int): (DecisionTreeModel, Double) = {
  val startTime = new DateTime()
  val model = DecisionTree.trainClassifier(trainData, 2, Map[Int, Int](),
    impurity, maxDepth, maxBins)
```

```
val endTime = new DateTime()

val duration = new Duration(startTime, endTime)

(model, duration.getMillis())

}
```

上述命令的详细说明如下：

➤ 定义 trainModel 函数

```
def trainModel(trainData: RDD[LabeledPoint], impurity: String,
    maxDepth: Int, maxBins: Int): (DecisionTreeModel, Double) = {
```

定义 trainModel function 传入下列参数：

trainData 训练数据、impurity 决策树的评估方法、maxDepth 决策树的最大深度、maxBins 决策树每一个节点的最大分支数目。

➤ 记录开始进行训练的时间

```
val startTime = new DateTime()
```

➤ 进行 DecisionTree 训练

```
val model = DecisionTree.trainClassifier(trainData, 2, Map[Int, Int](),
    impurity, maxDepth, maxBins)
```

进行 DecisionTree.trainClassifier 分类训练，并且传入参数 trainData、impurity、maxDepth、maxBins，完成后会产生模型 model。

上述命令中，最主要是以 MLlib 提供的方法 DecisionTree.trainClassifier 进行训练，介绍如下：

DecisionTree.trainClassifier(input, numClasses, categoricalFeaturesInfo, impurity, maxDepth, maxBins)		
返回	DecisionTreeModel	
参数	类型	说明
input	RDD[LabeledPoint]	输入的训练数据
numClasses	Int	分类数目
categoricalFeaturesInfo	Map[Int, Int]	是 Map[Int, Int]()
Impurity	String	决策树的 impurity 评估方法有 2 种方式： gini 基尼系数、entropy 熵
maxDepth	Int	决策树的最大深度
maxBins	Int	决策树每一个节点的最大分支数目

> 记录完成训练的时间

```
val endTime = new DateTime()
```

> 计算训练所需的时间

```
val duration = new Duration(startTime, endTime)
```

传入开始时间与结束时间，计算训练所需的时间。

> 返回数据

```
(model, duration.getMillis())
```

返回：模型与训练所需时间。

步骤 03 以 AUC (Area under the Curve of ROC) 评估数据模型

针对二元分类算法，主要是以 AUC (Area under the Curve of ROC) 来评估数据模型的好坏。

		Actual 真实值	
		positives	negatives
Predict 预测值	positives	True Positives (TP)	False Positives (FP)
	negatives	False Negatives (FN)	True Negatives (TN)

例如：

0 代表暂时性网页

1 代表长青网页

真阳性 True Positives (TP)：预测为 1；实际上为 1。

伪阳性 False Positives (FP)：预测为 1；实际上为 0。

真阴性 True Negatives (TN)：预测为 0；实际上为 1。

伪阴性 False Negatives (FN)：预测为 0；实际上为 0。

TPR：在所有实际为 1 的样本中，被正确地判断为 1 的比例。

$TPR=TP/(TP+FN)$

FPR：在所有实际为 0 的样本中，被错误地判断为 1 的比例。

$FPR=FP/(FP+TN)$

有了 TPR、FPR 就可以绘出 ROC 曲线图，如图 13-11 所示。AUC (Area under the Curve of ROC) 就是 ROC 曲线下的面积。

ROC 曲线图

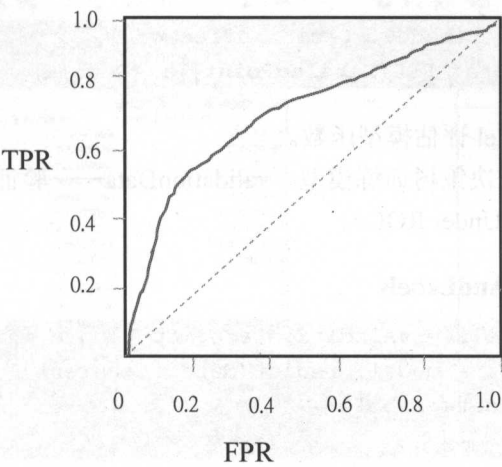


图 13-11 ROC 曲线图

可从 AUC 判断二元分类的优劣:

条件	说明
$AUC = 1$	是最完美的情况, 预测准确率 100%, 但是不可能存在
$0.5 < AUC < 1$	优于随机猜测, 具有预测价值
$AUC = 0.5$	与随机猜测一样, 没有预测价值
$AUC < 0.5$	比随机猜测还差, 但如果反向预测, 就优于随机猜测

步骤 04 评估模型 evaluateModel

上一步骤已经了解了 AUC 的概念, 接下来要在程序中使用 AUC 评估二元分类模型的好坏。计算 AUC 有些复杂, 不过不用担心, MLlib 已经有这个功能。MLlib 提供了使用 BinaryClassificationMetrics 计算 AUC, 使用方法如下:

输入下列程序:

```
def evaluateModel(model: DecisionTreeModel,
  validationData: RDD[LabeledPoint]): (Double) = {
  val scoreAndLabels = validationData.map { data =>
    var predict = model.predict(data.features)
    (predict, data.label)
  }
  val Metrics = new BinaryClassificationMetrics(scoreAndLabels)
  val AUC = Metrics.areaUnderROC
  (AUC)
}
```

上述命令的详细说明如下:

➤ 定义 `evaluateModel`

```
def evaluateModel(model: DecisionTreeModel,
  validationData: RDD[LabeledPoint]): (Double) = {
```

定义 `evaluateModel` 评估模型函数。

参数: `model`——决策树训练模型、`validationData`——验证数据。

返回 AUC (area Under ROC)。

➤ 创建 `scoreAndLabels`

```
val scoreAndLabels = validationData.map { data =>
  var predict = model.predict(data.features)
  (predict, data.label)
}
```

`scoreAndLabels` 是由 `score` (预测的结果) 与 `Labels` (真实的结果) 组成。

1. `validationData` 使用 `map` 将每一条数据进行转换。
2. `model.predict` 传入 `features` 参数进行预测, 得出预测结果 `predict`。
3. 返回 `scoreAndLabels` (`predict` 预测的结果、`data.label` 真实的结果)。

➤ 计算 AUC

```
val Metrics = new BinaryClassificationMetrics(scoreAndLabels)
val AUC = Metrics.areaUnderROC
```

MLlib 提供了计算 AUC 函数:

1. 先使用 `new BinaryClassificationMetrics` 传入参数 `scoreAndLabels` 创建 `Metrics`。
2. 有了 `Metrics` 之后, 只需使用 `Metrics.areaUnderROC` 方法就可得到 AUC (`areaUnderROC`)。

13.6 预测阶段

使用之前训练完成的模型, 并使用 `test.tsv` 进行预测。

步骤 01 介绍 `test.tsv`

我们将使用 `test.tsv` 进行预测, 先复习一下 `test.tsv`。可以看到 `test.tsv` 中没有 `Label` 字段, 其余字段和 `train.tsv` 完全相同, 如图 13-12 所示。

	X	Y	Z	AA	AB
1	numwords_in_url	parametrizedLinkRatio	spelling_errors_ratio		
2					
3					
4					
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					
16					
17					
18					
19					

图 13-12 test.tsv 中没有 Label 字段，其余字段和 train.tsv 完全相同

步骤 02 PredictData 程序代码

使用之前训练完成的模型，并使用 test.tsv 进行预测。

```
def PredictData(sc: SparkContext,
    model: DecisionTreeModel, categoriesMap: Map[String, Int]): Unit = {
  //-----1. 导入并转换数据-----
  val rawDataWithHeader = sc.textFile("data/test.tsv")
  val rawData = rawDataWithHeader
    .mapPartitionsWithIndex { (idx, iter) => if (idx == 0) iter.drop(1) else iter }
  val lines = rawData.map(_.split("\t"))
  println("共计: " + lines.count.toString() + "条")
  //-----2. 创建训练评估所需的数据 RDD[LabeledPoint]-----
  val dataRDD = lines.take(20).map { fields =>
    val trFields = fields.map(_.replaceAll("\\", ""))
    val categoryFeaturesArray = Array.ofDim[Double](categoriesMap.size)
    val categoryIdx = categoriesMap(fields(3))
    categoryFeaturesArray(categoryIdx) = 1
    val numericalFeatures = trFields.slice(4, fields.size)
      .map(d => if (d == "?") 0.0 else d.toDouble)
    val label = 0
  }
  //-----3. 进行预测-----
  val url = trFields(0)
  val Features = Vectors.dense(categoryFeaturesArray ++ numericalFeatures)
  val predict = model.predict(Features).toInt
  var predictDesc = { predict match {
    case 0 => "暂时性网页(ephemeral)";
    case 1 => "长青网页(evergreen)"; } }
  println(" 网址: " + url + "==>预测:" + predictDesc)
}
```


PredictData 与 PrepareData 程序代码类似，以下仅说明不同之处。

上述命令的详细说明如下：

➤ 定义 PredictData

```
def PredictData(sc: SparkContext,
               model: DecisionTreeModel, categoriesMap: Map[String, Int]): Unit = {
```

PredictData 传入下列参数：sc——SparkContext、model——之前训练完成的模型、categoriesMap——传入之前 PrepareData（数据准备阶段）时创建的对照表；预测时也必须使用此对照表。对照表必须相同，才不会发生错误。

如果 PrepareData 与 PredictData 使用各自生成的对照表，会造成下列情况：

- PrepareData 字段 3 “分类特征字段”在 train.tsv 中的网页分类内有 N 个分类，就会转换为 N 个数值字段。
- PredictData 字段 3 “分类特征字段”在 test.tsv 中的网页分类内有 M 个分类，就会转换为 M 个数值字段。

因为特征字段数不同，会造成在执行预测时发生错误。

➤ 读取 test.tsv

```
val rawDataWithHeader = sc.textFile("data/test.tsv")
```

使用 sc.textFile 读取 test.tsv。

➤ 处理每一行数据

```
val dataRDD = lines.take(10).map { fields =>
  ...
}
```

使用 map 针对每一条数据进行处理，传入 fields 作为参数。test.tsv 共有 3000 多条数据，我们在这里只读取 10 条进行预测。

➤ 字段 4~25 是 numericalFeatures 数值特征字段

```
val numericalFeatures = trFields.slice(4, fields.size)
    .map(d => if (d == "?") 0.0 else d.toDouble)
```

在 PrepareData 中，我们读取 train.tsv 使用 slice(4, r.size - 1) 提取字段，这是因为最后一个字段是 label。但是在 PredictData 读取 test.tsv 时，因为没有 label 字段，所以我们使用 .slice(4, fields.size) 提取字段。

➤ label 字段

```
val label = 0
```


因为 test.tsv 没有 label 字段，所以设置 val label = 0。

➤ 读取第 0 个字段 url 网址

```
val url = trFields(0)
```

读取第 0 个字段 url 网址，因为后续希望能显示网址与预测结果。

➤ 进行预测

```
val Features = Vectors.dense(categoryFeaturesArray ++ numericalFeatures)
```

Features 是由 categoryFeaturesArray ++ numericalFeatures 数组相加而成。

```
val predict = model.predict(Features).toInt
```

使用 model.predict 传入 Features 参数进行预测，预测结果存储于 predict。

➤ 显示预测结果

```
var predictDesc = { predict match {  
  case 0 => "暂时性网页(ephemeral)";  
  case 1 => "长青网页(evergreen)"; } }
```

预测结果 predict 是 0 或 1，以上列命令转换为文字描述。0 代表暂时性网页、1 代表长青网页。

```
println("网址: " + url + "==>预测:" + predictDesc)
```

显示网址与预测结果。

13.7 运行 RunDecisionTreeBinary.scala

步骤 01 依次选择菜单及其菜单项 Run→Run Configurations（见图 13-13）

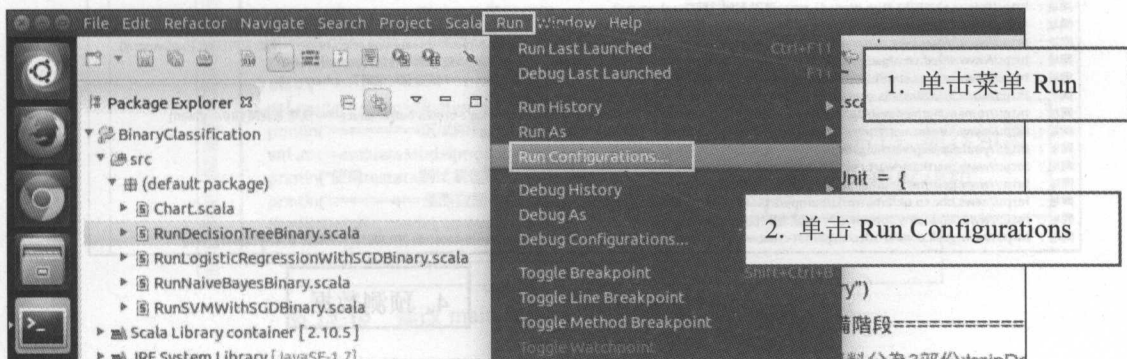


图 13-13 依次选择菜单及其菜单项 Run→Run Configurations

步骤 02 设置 Run Configurations (见图 13-14)

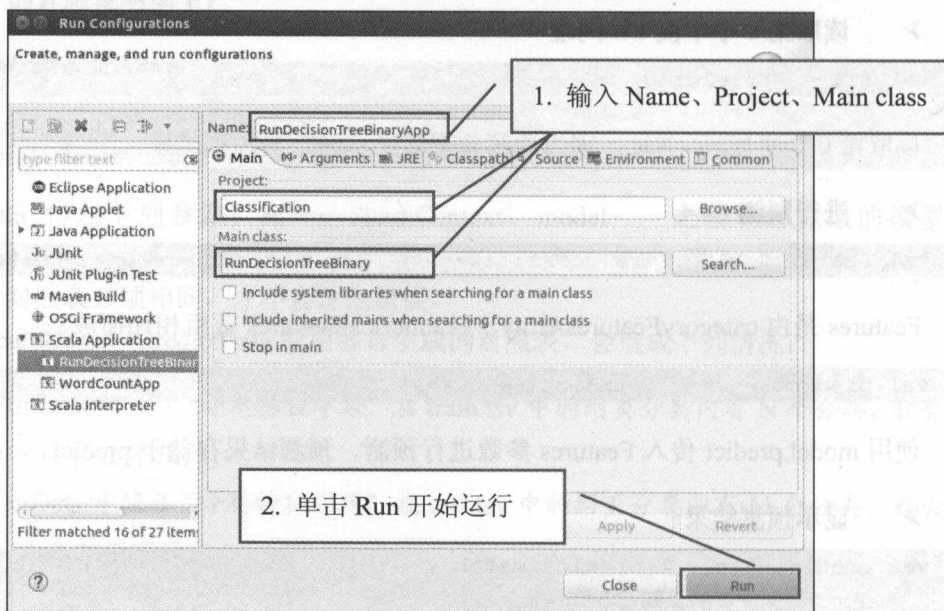


图 13-14 设置 Run Configurations (运行配置)

步骤 03 运行 RunDecisionTreeBinary 的结果

运行后屏幕显示界面如图 13-15 所示。

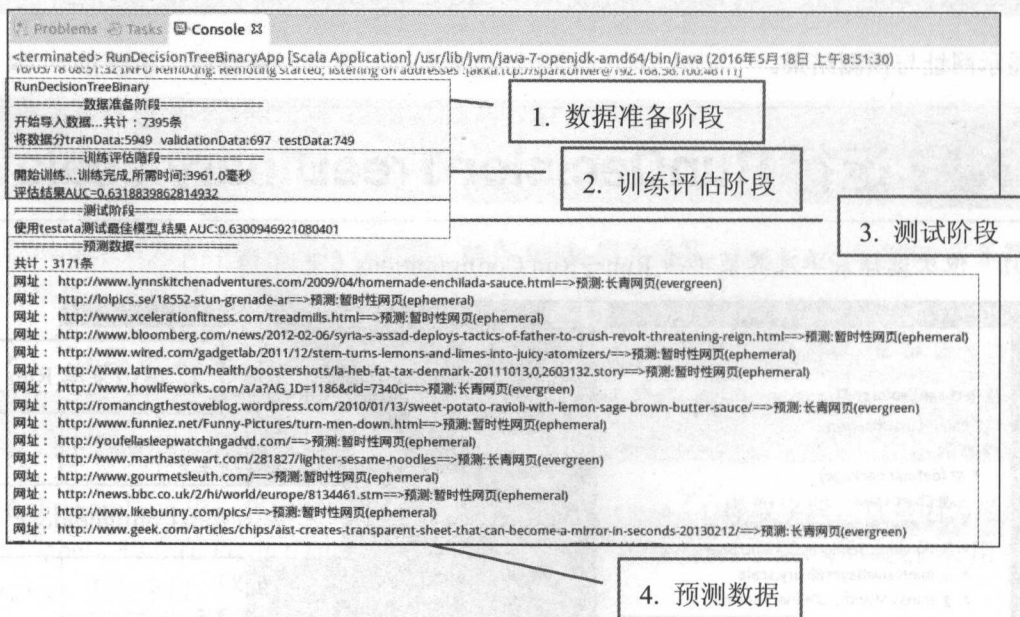


图 13-15 运行 RunDecisionTreeBinary 的结果

运行结果说明如下：

1. **数据准备阶段**——显示导入个数以及 trainData validationData testData 个数。
2. **训练评估阶段**——显示训练所需时间，以及评估结果 AUC（大约 0.6）。
3. **测试阶段**——使用测试数据再测试模型。AUC 大约 0.64，与训练阶段差异不大，确认没有 overfitting 的问题。
4. **预测阶段**——使用 test.tsv 预测是否为长青网页或暂时性网页。预测的结果显示了网址，可以使用浏览器查看此网址，检查预测的结果与人工判断是否相符。

13.8

修改 RunDecisionTreeBinary 调校训练参数

在之前的章节中，我们使用 trainClassifier 命令进行训练，会返回 model 训练完成的模型。

```
DecisionTree.trainClassifier(input, numClasses, categoricalFeaturesInfo,
impurity, maxDepth, maxBins)
```

其中 impurity、maxDepth、maxBins 这些参数值的设置，会影响结果的准确度以及训练所需的时间。接下来，将进行调校找出最佳的参数组合。

步骤 01 修改 main function（见图 13-16）

```
RunDecisionTreeBinary.scala ✕

def main(args: Array[String]): Unit = {
  SetLogger()
  val sc = new SparkContext(new SparkConf().setAppName("App").setMaster("local[4]"))
  println("RunDecisionTreeBinary")
  println("=====数据准备阶段=====")
  val (trainData, validationData, testData, categoriesMap) = PrepareData(sc)
  trainData.persist(); validationData.persist(); testData.persist()
  println("=====训练评估阶段=====")
  println()
  print("是否需要進行参数調校 (Y:是 N:否) ? ")
  if (readLine() == "Y") {
    val model = parametersTuning(trainData, validationData)
    println("=====测试阶段=====")
    val auc = evaluateModel(model, testData)
    println("使用testata测试最佳模型,结果 AUC:" + auc)
    println("=====预测数据=====")
    PredictData(sc, model, categoriesMap)
  } else {
    val model = trainEvaluate(trainData, validationData)
    println("=====测试阶段=====")
    val auc = evaluateModel(model, testData)
    println("使用testata测试最佳模型,结果 AUC:" + auc)
    println("=====预测数据=====")
    PredictData(sc, model, categoriesMap)
  }
}
```

修改程序，加入如图
所示的程序代码

图 13-16 修改 main function 中的程序，加入新的代码

以上程序会询问是否要进行参数调校，输入 Y 或不是 Y。

输入	执行
Y	<code>val model = parametersTunning(trainData, validationData)</code> 进行参数调校
不是 Y	<code>val model = trainEvaluate(trainData, validationData)</code> 进行训练评估

训练与评估在之前章节已经介绍过，因此省略。以下将介绍进行参数调校。

步骤02 输入 parametersTunning 参数调校函数

输入 parametersTunning 函数如下：

```
def parametersTunning(trainData: RDD[LabeledPoint],
  validationData: RDD[LabeledPoint]): DecisionTreeModel = {
  println("-----评估 Impurity 参数使用 gini, entropy-----")
  evaluateParameter(trainData, validationData, "impurity",
    Array("gini", "entropy"), Array(10), Array(10))
  println("-----评估 MaxDepth 参数使用 (3, 5, 10, 15, 20)-----")
  evaluateParameter(trainData, validationData, "maxDepth",
    Array("gini"), Array(3, 5, 10, 15, 20, 25), Array(10))
  println("-----评估 maxBins 参数使用 (3, 5, 10, 50, 100)-----")
  evaluateParameter(trainData, validationData, "maxBins",
    Array("gini"), Array(10), Array(3, 5, 10, 50, 100, 200))
  println("-----所有参数交叉评估找出最好的参数组合-----")
  val bestModel = evaluateAllParameter(trainData, validationData,
    Array("gini", "entropy"), Array(3, 5, 10, 15, 20), Array(3, 5, 10, 50, 100))
  return (bestModel)
}
```

在这里评估的参数共有 3 个：Impurity、maxDepth、maxBins。我们希望得知不同的参数值对准确率的影响，以及运行所需要的时间。评估的方法是同一时间只评估一个参数。

➤ 评估 Impurity 参数

我们要评估 Impurity 参数的方法是固定 maxdepth= Array(10)、maxBins= Array(10)。但是 impurity 输入参数有 2 个不同值 Array ("gini", "entropy"), 用于评估哪一个参数值具有比较好的准确率——也就是 AUC 值比较高。

命令	说明
<code>evaluateParameter(trainData, validationData, "impurity", Array("gini", "entropy"), Array(10), Array(10))</code>	<code>evaluateParameter</code> 函数传入下列参数: trainData、validationData; 设置当前的是评估 impurity 参数; impurity 输入参数有 2 个不同值("gini", "entropy"); maxdepth 固定是 10; maxBins 固定是 10

➤ 评估 MaxDepth 参数

当我们要评估 maxdepth 参数时，固定 impurityArray =Array("gini")、maxBinsArray = Array(10)。但是 maxdepthArray=Array(3, 5, 10, 15, 20, 25)传入 maxdepthArray 值来评估哪一个参数具有比较好的准确率——也就是 AUC 值比较高。

命令	说明
<pre>evaluateParameter(trainData,validationData, "maxdepth", Array("gini"), Array(3, 5, 10, 15, 20, 25), Array(10))</pre>	<p>evaluateParameter 函数传入下列参数：</p> <p>trainData、validationData;</p> <p>设置当前的是评估 maxdepth 参数;</p> <p>impurity 固定是"gini";</p> <p>maxdepth 输入参数有 6 个不同值: 3、5、10、15、20、25;</p> <p>maxBins 固定是 10</p>

➤ 评估 maxBins 参数

当我们要评估 maxBins 参数的方法是固定 impurity= Array("gini")、maxdepth= Array(10)。但是 maxBins 输入参数有 6 个不同值 Array (3, 5, 10, 50, 100, 200)，用来评估哪一个参数具有比较好的准确率——也就是 AUC 值比较高。

命令	说明
<pre>evaluateParameter(trainData,validationData, "maxBins", Array("gini"), Array(10), Array(3, 5, 10, 50, 100, 200))</pre>	<p>evaluateParameter 函数传入下列参数：</p> <p>trainData、validationData;</p> <p>设置当前是评估 maxBins 参数;</p> <p>impurity 固定是"gini";</p> <p>maxdepth 固定是 10;</p> <p>maxBins 输入参数有 6 个不同值 3、5、10、50、100、200</p>

➤ 所有参数交叉评估

最后，程序中有关所有参数交叉评估找出最好的参数组合，会在后续章节说明。

```
println("-----所有参数交叉评估找出最好的参数组合-----")  
val bestModel = evaluateAllParameter(trainData, validationData,  
  Array("gini", "entropy"),Array(3, 5, 10, 15, 20, 25),  
  Array(3, 5, 10, 50, 100, 200))  
return (bestModel)
```

步骤 03 输入 evaluateParameter 函数

此函数主要是评估单个参数，看哪一个参数值具有比较好的准确率，并且画出图形。

```
def evaluateParameter(trainData: RDD[LabeledPoint],
```

```

validationData: RDD[LabeledPoint], evaluateParameter: String,
impurityArray: Array[String], maxdepthArray: Array[Int],
maxBinsArray: Array[Int]) =
{
  var dataBarChart = new DefaultCategoryDataset()
  var dataLineChart = new DefaultCategoryDataset()
  for (impurity <- impurityArray;
      maxDepth <- maxdepthArray;
      maxBins <- maxBinsArray) {
    val (model, time) = trainModel(trainData, impurity, maxDepth, maxBins)
    val auc = evaluateModel(model, validationData)
    val parameterData =
      evaluateParameter match {
        case "impurity" => impurity;
        case "maxDepth" => maxDepth;
        case "maxBins" => maxBins
      }
    dataBarChart.addValue(auc, evaluateParameter,
parameterData.toString())
    dataLineChart.addValue(time, "Time", parameterData.toString())
  }
  Chart.plotBarLineChart("DecisionTree evaluations " +
    evaluateParameter, evaluateParameter, "AUC",
    0.58, 0.7, "Time", dataBarChart, dataLineChart)
}

```

上述命令的详细说明如下:

➤ 定义 evaluateParameter 函数

```

def evaluateParameter(trainData: RDD[LabeledPoint],
validationData: RDD[LabeledPoint], evaluateParameter: String,
impurityArray: Array[String], maxdepthArray: Array[Int],
maxBinsArray: Array[Int]) =

```

evaluateParameter 函数传入下列参数: trainData 训练数据、validationData 验证数据、impurity 参数、maxdepth 参数、maxBins 参数。

➤ 创建绘图数据集

```
var dataBarChart = new DefaultCategoryDataset()
```

创建柱形图所需要的数据集。

```
var dataLineChart = new DefaultCategoryDataset()
```

创建折线图所需要数据集。

➤ 评估参数

```
for (rank <- rankArray;
```

```
numIterations <- numIterationsArray;
lambda <- lambdaArray) {
```

for 命令会重复执行 Array 中的每一个值。例如，要评估 rank 参数的方法是固定 numIterations = Array(10)、lambda = Array(0.1)。但是 rank 参数有 6 个不同值 Array(5, 10, 15, 20, 0, 100)，所以程序会执行 6 次，以便评估哪一个参数值具有较高 AUC。

➤ 执行训练评估

```
val (model, time) = trainModel(trainData, impurity, maxDepth, maxBins)
```

trainModel 执行训练：

- 传入参数：trainData（训练数据）、impurity、maxDepth、maxBins。
- 返回：此训练完成的模型、训练所需时间。

```
val auc = evaluateModel(model, validationData)
```

evaluateModel 进行评估

- 传入参数：训练完成的模型，validationData（验证数据）。
- 返回：auc。

➤ 设置绘图数据

```
dataBarChart.addValue(auc, evaluateParameter, parameterData.toString())
```

加入柱形图数据，柱形图用于显示 AUC。

```
dataLineChart.addValue(time, "Time", parameterData.toString())
```

加入折线图数据，折线图用于显示执行训练所需的时间。

➤ 开始绘图

```
Chart.plotBarLineChart("DecisionTree evaluations " +
  evaluateParameter, evaluateParameter, "AUC",
  0.58, 0.7, "Time", dataBarChart, dataLineChart)
```

绘制柱形图与折线图。

步骤 04 evaluateAllParameter

evaluateAllParameter 程序将 3 个参数交叉评估，以找出最好的参数组合。请输入 evaluateAllParameter 函数程序代码如下：

```
def evaluateAllParameter(trainData: RDD[LabeledPoint],
  validationData: RDD[LabeledPoint],
  impurityArray: Array[String],
```



```
maxdepthArray: Array[Int],
maxBinsArray: Array[Int]): DecisionTreeModel =
{
    val evaluationsArray =
        for (impurity <- impurityArray;
maxDepth <- maxdepthArray;
maxBins <- maxBinsArray) yield {
            val (model, time) = trainModel(trainData, impurity, maxDepth, maxBins)
            val auc = evaluateModel(model, validationData)
            (impurity, maxDepth, maxBins, auc)
        }
    val BestEval = (evaluationsArray.sortBy(_._4).reverse)(0)
    println("调校后最佳参数: impurity:" + BestEval._1 +
" ,maxDepth:" + BestEval._2 +
" ,maxBins:" + BestEval._3 +
" ,结果 AUC = " + BestEval._4)

    val (bestModel, time) = trainModel(
trainData.union(validationData), BestEval._1, BestEval._2, BestEval._3)
    return bestModel
}
```

调用 evaluateAllParameter 函数的方式如下:

命令	说明
<pre>val bestModel = evaluateAllParameter(trainData, validationData, Array("gini", "entropy"), Array(3, 5, 10, 15, 20, 25), Array(3, 5, 10, 50, 100, 200))</pre>	<p>evaluateAllParameter 函数执行后将返回最佳的模型, 传入下列参数:</p> <p>trainData、validationData;</p> <p>Impurity 输入参数有 2 个不同值;</p> <p>maxdepth 输入参数有 6 个不同值;</p> <p>maxBins 输入参数有 6 个不同值;</p> <p>运行后返回最佳的模型</p>

以上 3 个参数共有 2*6*6=72 个排列组合。我们会对每一种参数组合评估 AUC, 最后找出具有最大的 AUC 的参数组合就是最佳组合。

evaluateAllParameter 程序代码的详细说明如下:

➤ 定义 evaluateAllParameter 函数

```
def evaluateAllParameter(trainData: RDD[LabeledPoint],
validationData: RDD[LabeledPoint],
impurityArray: Array[String],
maxdepthArray: Array[Int],
maxBinsArray: Array[Int]): DecisionTreeModel =
```


evaluateAllParameter 函数:

- 传入参数: trainData 训练数据、validationData 验证数据、impurityArray 参数、maxdepthArray 参数、maxBinsArray 参数。
- 返回 DecisionTreeModel 模型。

➤ for 循环交叉评估所有的参数组合

```
val evaluationsArray =
  for (impurity <- impurityArray;
      maxDepth <- maxdepthArray;
      maxBins <- maxBinsArray) yield {
    val (model, time) = trainModel(trainData, impurity, maxDepth, maxBins)
    val auc = evaluateModel(model, validationData)
    println("参数: impurity:" + impurity +
      " ,maxDepth:" + maxDepth +
      " ,maxBins:" + maxBins +
      " ,结果 AUC = " + auc)
    (impurity, maxDepth, maxBins, auc)
  }
```

1. for 循环有 3 个数组: impurityArray、maxdepthArray、maxBinsArray, 3 个 Array 排列组合, 共有 $2*6*6=72$ 种排列组合。
2. 因为加入了 yield 关键词, 每运行 1 次会产生 1 条数据。
3. 使用 trainModel 训练模型与 evaluateModel 评估模型。
4. 最后会产生 72 种排列组合 Tuple (impurityArray、maxdepthArray、maxBinsArray、auc), 所组成的数组存入 evaluationsArray。

➤ 找出 AUC 最大的参数组合

```
val BestEval = (evaluationsArray.sortBy(_._4).reverse)(0)
```

之前已产生 72 种排列组合的数组 evaluationsArray, 所以我们在 72 条参数组合中, 找出 AUC 中最大的参数组合。

1. 使用 “.sortBy(_._4)” 用 evaluationsArray 的第 4 个字段, 也就是 AUC 进行排序。
2. “.reverse” 从大到小排序。
3. 使用 “(0)” 取出排序后的第一条数据, 存入 BestEval 中就是最佳参数组合。

➤ 显示最佳参数组合与 AUC

```
println("调校后最佳参数: impurity:" + BestEval._1 +
  " ,maxDepth:" + BestEval._2 +
  " ,maxBins:" + BestEval._3 +
  " ,结果 AUC = " + BestEval._4)
```

BestEval 是 Tuple (impurity, maxdepth, maxBins, auc), 在 scala 中可以使用下列语句读取每个元素:

BestEval._1 读取第 1 个元素 impurity
 BestEval._2 读取第 2 个元素 maxDepth
 BestEval._3 读取第 3 个元素 maxBins
 BestEval._4 读取第 4 个元素 auc

➤ 使用最佳参数组合进行训练

```
val (bestModel, time) = trainModel(
  trainData.union(validationData), BestEval._1, BestEval._2, BestEval._3)
```

ALS.train 使用最佳参数组合进行训练:

1. 参数传入——训练数据 trainData.union(validationData)。
2. 最佳组合参数——impurity、maxdepth、maxBins。
3. 返回 bestModel 最佳模型与训练所需的时间。

13.9

运行 RunDecisionTreeBinary 进行参数调校

接下来我们要运行参数调校程序, 并且针对每一个参数绘制出柱形图与折线图。在此必须先说明, 因为我们的 trainData、validationData、testData 是随机产生的, 所以每次运行的结果可能不同。因此, 运行的结果可能与书上的运行结果不同, 不过差异不会太大。

步骤 01 运行程序 (见图 13-17)

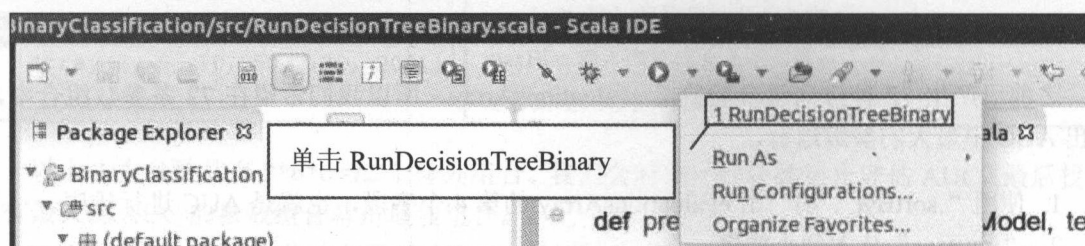


图 13-17 RunDecisionTreeBinary 程序

步骤 02 询问是否进行参数调校

程序会询问是否需要参数调校 (Y: 是、N: 否), 请输入 Y。如图 13-18 所示。

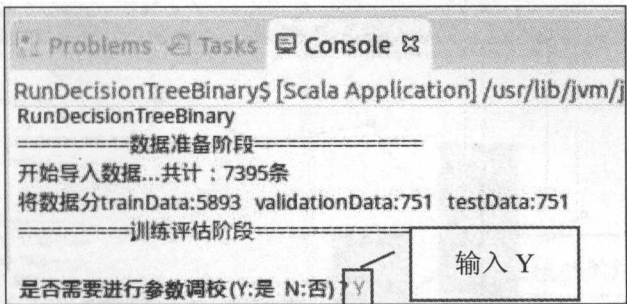


图 13-18 选择进行参数调校

步骤 03 Impurity 参数评估

接下来，程序会显示如图 13-19 所示。

柱形图代表 AUC、折线图代表运行的时间。从下图可以看到 entropy 准确度比 gini 好一些，但是并没有很大的差别。

运行时间：entropy 所需的时间只有 gini 的 1/4。

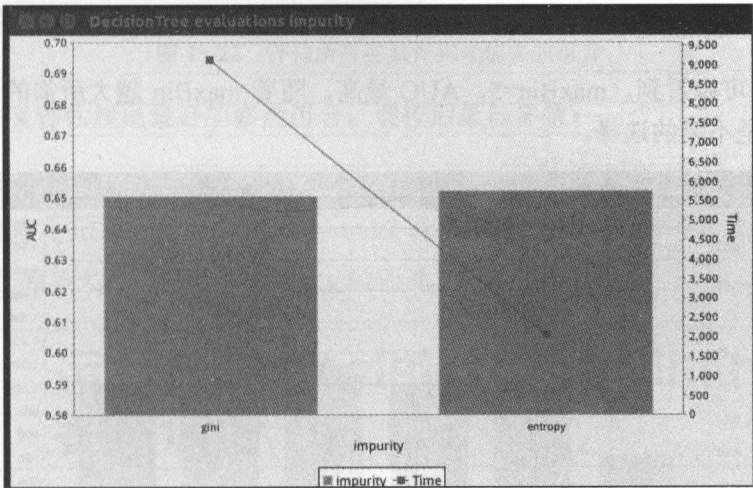


图 13-19 Impurity 参数评估结果图

步骤 04 maxDepth 参数评估

从图 13-20 可以看到，maxDepth=5，AUC 最高，随着 maxDepth 越大所需的时间越多。所以 maxDepth=5 可能是不错的选择。

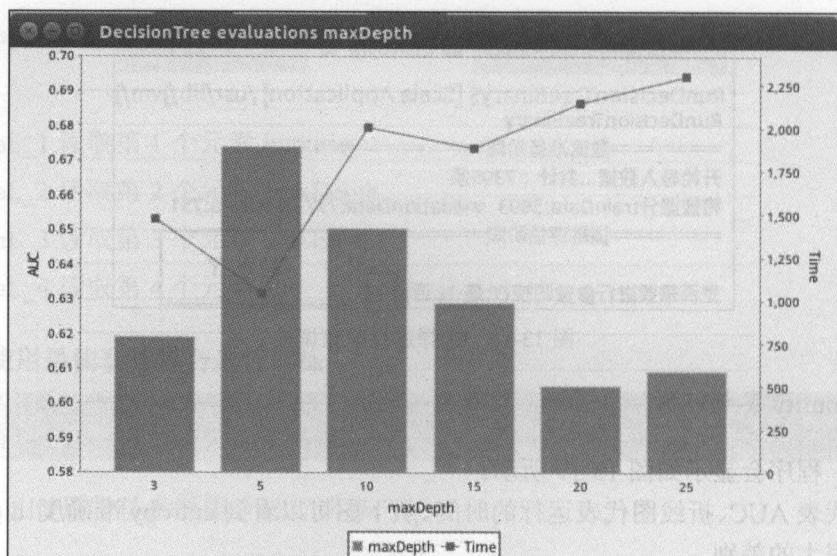


图 13-20 maxDepth 参数评估结果图

步骤 05 maxBins 参数评估

从图 13-21 可以看到, maxBin=5, AUC 最高, 随着 maxBin 越大所需的时间越多。所以 maxBin=5 可能是不错的选择。

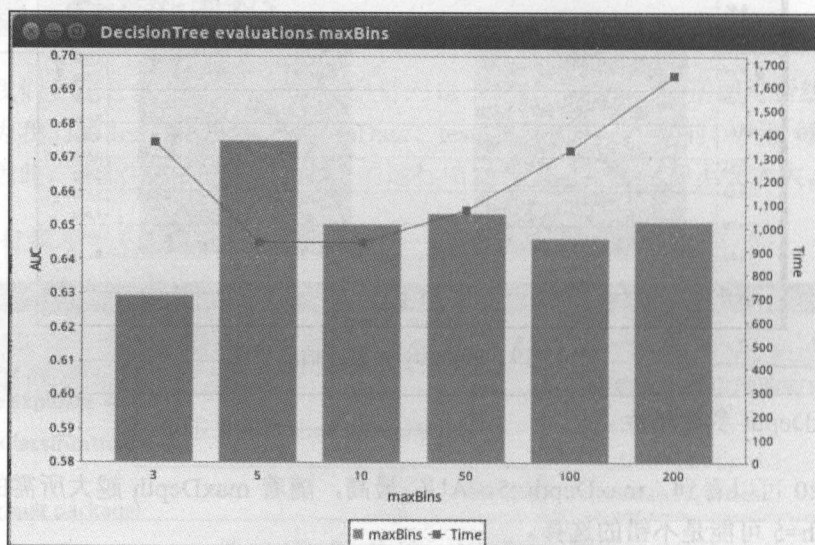


图 13-21 maxBins 参数评估结果图

步骤 06 评估所有参数

最后程序运行界面如图 13-22 所示, 找出最佳参数组合:


```

<terminated> RunDecisionTreeBinaryApp [Scala Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (2016年5月18日 上午9:19:59)
RunDecisionTreeBinary
=====数据准备阶段=====
开始导入数据...共计: 7395条
将数据分trainData:5919 validationData:746 testData:730
=====训练评估阶段=====
是否需要参数调校 (Y:是 N:否)? Y
--评估 Impurity 参数使用 gini, entropy -----
--评估 MaxDepth 参数使用 (3, 5, 10, 15, 20) -----
--评估 maxBins 参数使用 (3, 5, 10, 50, 100) -----
所有参数交叉评估找出最好的参数组合
调校后最佳参数: impurity:gini, maxDepth:10, maxBins:10, 结果AUC = 0.6876345371920594
=====测试阶段=====
使用testata测试最佳模型, 结果 AUC:0.6600483425414365
=====预测数据=====
共计: 3171条
网址: http://www.lynnskitchenadventures.com/2009/04/homemade-enchilada-sauce.html==>预测: 长青网页(evergreen)
网址: http://lolpics.se/18552-stun-grenade-ar==>预测: 暂时性网页(ephemeral)
网址: http://www.xcelerationfitness.com/treadmills.html==>预测: 暂时性网页(ephemeral)
网址: http://www.bloomberg.com/news/2012-02-06/syria-s-assad-deploys-tactics-of-father-to-crush-revolt-threatening-reign.html==>预测: 暂时性网页(ephemeral)
网址: http://www.wired.com/gadgetlab/2011/12/stem-turns-lemons-and-limes-into-juicy-atomizers==>预测: 暂时性网页(ephemeral)
网址: http://www.latimes.com/health/boostershots/la-heb-fat-tax-denmark-20111013,0,2603132.story==>预测: 暂时性网页(ephemeral)
网址: http://www.howlifeworks.com/a/a7AG_ID=1186&cid=7340ci==>预测: 暂时性网页(ephemeral)
网址: http://romancingthetoveblog.wordpress.com/2010/01/13/sweet-potato-ravioli-with-lemon-sage-brown-butter-sauce/=>预测: 长青网页(evergreen)
网址: http://www.funniez.net/Funny-Pictures/turn-men-down.html==>预测: 暂时性网页(ephemeral)
网址: http://youfellasleepwatchingadvd.com/=>预测: 长青网页(evergreen)
网址: http://www.marthastewart.com/281827/lighter-sesame-noodles==>预测: 长青网页(evergreen)
网址: http://www.gourmetsleuth.com/=>预测: 暂时性网页(ephemeral)
网址: http://news.bbc.co.uk/2/hi/world/europe/8134461.stm==>预测: 暂时性网页(ephemeral)

```

图 13-22 评估所有参数得到最佳参数组合

所有参数交叉评估找出最好的参数组合，调校后最佳参数：

- 最佳 model 的参数组合：impurity:gini、maxDepth:10、maxBins:10，结果 AUC = 0.68。
- 最后使用 testData 再次测试 bestModel 结果仍有 0.66，测试与训练评估阶段其结果 AUC 差异不大，代表无 overfitting 的问题。

13.10

运行 RunDecisionTreeBinary 不进行参数调校

因为参数调校比较花费时间。当我们已经找出最佳参数组合，则可以直接将 trainEvaluate 程序改为最佳参数组合。下次要运行预测程序时，可以不需要再运行参数调校，就可直接使用最佳参数进行预测。

步骤 01 修改 trainEvaluate 为最佳参数组合

修改 trainEvaluate 程序，并改为最佳参数组合，如图 13-23 所示。

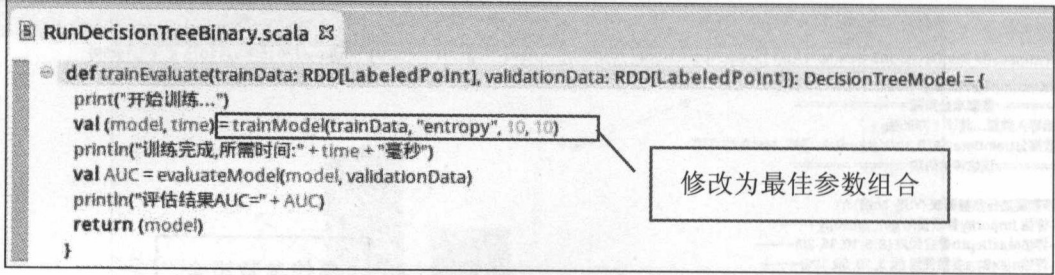


图 13-23 修改 trainEvaluate 为最佳参数组合

步骤 02 再次运行程序（见图 13-24）

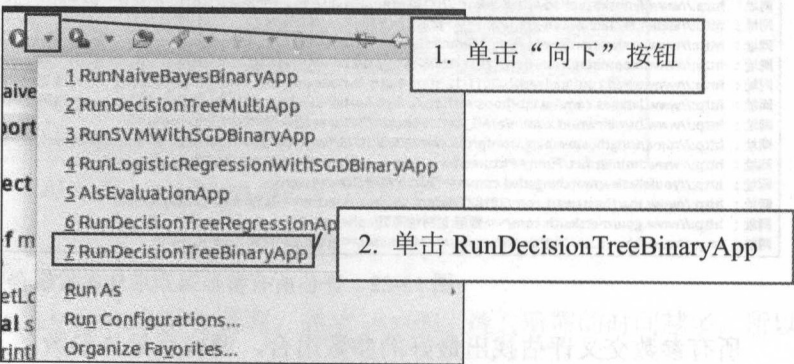


图 13-24 再次运行 RunDecisionTreeBinaryApp

步骤 03 是否需要进行参数调校

再次运行程序，程序会询问是否需要进行参数调校（Y：是、N：否），输入 N，如图 13-25 所示。

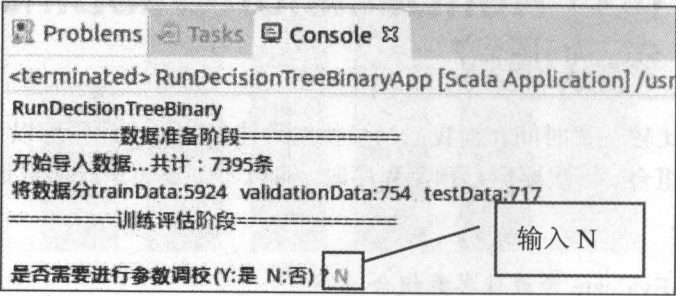


图 13-25 这次选择不进行参数调校

步骤 04 再次运行程序

运行结果如图 13-26 所示。可以看到程序训练完成之后，就可以进行预测。因为不进行参数调校，所以运行所需的时间比较少。

```
Problems Tasks Console
<terminated> RunDecisionTreeBinaryApp [Scala Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (2016年5月18日 上午9:26:42)
RunDecisionTreeBinary
=====数据准备阶段=====
开始导入数据...共计: 7395条
将数据分trainData:5924 validationData:754 testData:717
=====训练评估阶段=====
是否需要调参(Y/N)? N
开始训练...训练完成,所需时间:5843.0毫秒
评估结果AUC=0.6581386490604367
=====测试阶段=====
使用testata测试最佳模型,结果 AUC:0.6215508520737687
=====预测数据=====
共计: 3171条
网址: http://www.lynnskitchenadventures.com/2009/04/homemade-enchilada-sauce.html==>预测:暂时性网页(ephemeral)
网址: http://olpics.se/18552-stun-grenade-ar==>预测:暂时性网页(ephemeral)
网址: http://www.xcelerationfitness.com/treadmills.html==>预测:暂时性网页(ephemeral)
网址: http://www.bloomberg.com/news/2011-02-06/syria-s-assad-deploys-tactics-of-father-to-crush-revolt-threaten-reign.html==>预测:暂时性网页(ephemeral)
网址: http://www.wired.com/gadgetlab/2011/12/stem-turns-lemons-and-limes-into-juliy-atomizers/=>预测:暂时性网页(ephemeral)
网址: http://www.latimes.com/health/boostershots/la-heb-fat-tax-denmark-20111013_0_2603132.story==>预测:长青网页(evergreen)
网址: http://www.howlileworks.com/a/a?AG_ID=1186&cid=7340d==>预测:长青网页(evergreen)
网址: http://romancingthetoveblog.wordpress.com/2010/01/13/sweet-potato-ravioli-with-lemon-sage-brown-butter-sauce/=>预测:长青网页(evergreen)
网址: http://www.funniez.net/Funny-Pictures/turn-men-down.html==>预测:暂时性网页(ephemeral)
网址: http://youfellasleepwatchingadvd.com/=>预测:长青网页(evergreen)
```

图 13-26 选择最佳参数组合且不进行参数调校的预测程序的运行结果

第 14 章

逻辑回归二元分类

- 14.1 逻辑回归分析介绍
- 14.2 RunLogisticRegression WithSGDBinary.scala 程序说明
- 14.3 运行 RunLogisticRegression WithSGDBinary.scala 进行参数调校
- 14.4 运行 RunLogisticRegression WithSGDBinary.scala 不进行参数调校

本章中将使用第 12 章介绍的 StumbleUpon 数据集，运用逻辑回归二元分类来预测网页是暂时性的或是长青的，并且进行参数调校找出最佳参数组合，从而提高预测准确度。

14.1 逻辑回归分析介绍

➤ 简单回归分析 (Simple Regression Analysis)

先看看简单线性回归 (Simple Linear Regression)。它是假设变量 y 的数值，是自变量 x 所组成的某种线性函数值，而再加上一个误差值所得到的数值，如以下的数学公式：

$$y = b_0 + b_1x$$

例如：年龄与疾病的关系是线性回归的例子。随着年龄的增长，得到某疾病的机会也增加，如图 14-1 所示。

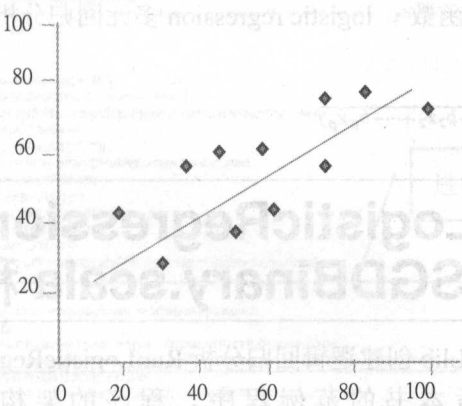


图 14-1 线性回归的图形示例

在线性回归中“因变量”是连续变项。如果“因变量”不是连续变项，而是二分变项，例如：是否得病。这时就必须使用逻辑回归分析 (Logistic Regression) 了。在逻辑回归分析中，我们可以将线性回归的公式：

$$y = b_0 + b_1x$$

转换为 sigmoid 的函数，来帮助界定某个 data 的类。该函数如下：

$$p = \frac{1}{1 + e^{-(b_0 + b_1x)}}$$

通过 sigmoid 计算出来的值 p (probability) 如果大于 0.5，则归类为“会得病”；反之则归类为“不会得病”。逻辑回归的图形示例如图 14-2 所示。

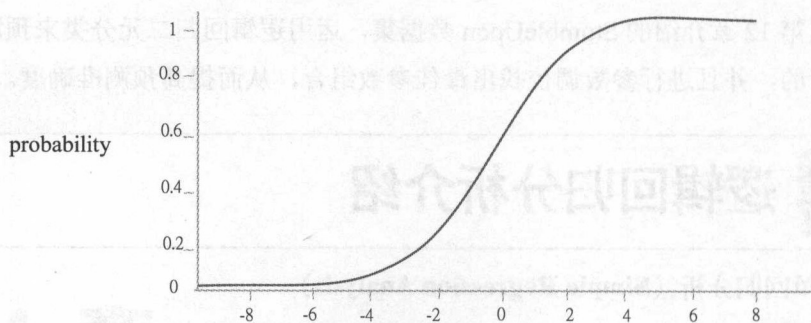


图 14-2 线性回归的图形示例

► 多元回归分析 (Multiple Regression Analysis, 或称为复回归分析)

以上是简单线性回归只使用一个自变量 x , 多元回归分析使用超过一个自变量, 公式如下:

$$y = b_0 + b_1x_1 + b_2x_2 + \dots + b_px_p$$

将其转换为 sigmoid 的函数, logistic regression 多元回归分析的公式如下:

$$p = \frac{1}{1 + e^{-(b_0 + b_1x_1 + b_2x_2 + \dots + b_px_p)}}$$

14.2

RunLogisticRegression WithSGDBinary.scala 程序说明

我们已经使用 Spark MLlib 创建逻辑回归分析 RunLogisticRegressionWithSGDBinary.scala, 完整的程序代码可参考本书的范例程序。程序的架构与之前第 13 章介绍的 RunDecisionTreeBinary.scala 类似, 可以参考 13 章的详细说明。本章仅说明 RunLogisticRegressionWithSGDBinary.scala 重要的修改部分。

步骤 01 导入 LogisticRegression 链接库 (见图 14-3)

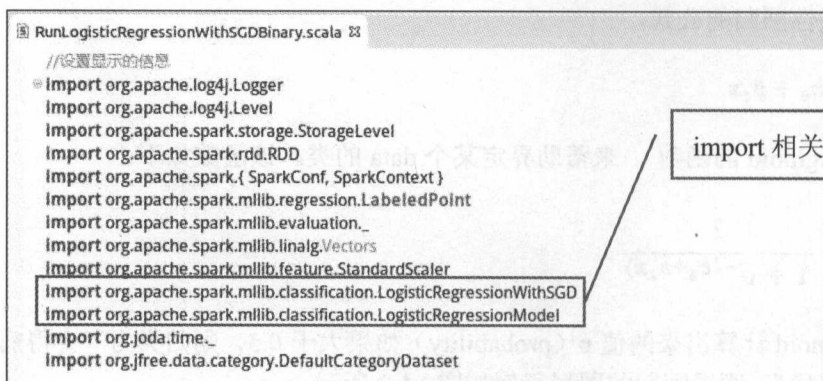


图 14-3 导入 LogisticRegression 链接库

首先必须先导入 LogisticRegression 相关链接库。

```
import org.apache.spark.mllib.classification.LogisticRegressionWithSGD
import org.apache.spark.mllib.classification.LogisticRegressionModel
```

步骤 02 修改 PrepareData() 加入进行数据标准化的程序代码

我们仍将使用第 12 章 StumbleUpon 数据集来执行逻辑回归分析二元分类。

下列程序基本上与第 13 章决策树的数据准备 PrepareData() 类似。唯一不同的是,当我们进行回归分析算法时,必须将数值特征 (Numerical Features) 字段进行标准化。因为数值特征字段单位不同,数字差异很大,所以彼此无法比较。这时就要使用标准化,让数值特征字段具有共同的标准。

修改 PrepareData(), 加入进行数据标准化的程序代码, 如图 14-4 所示。

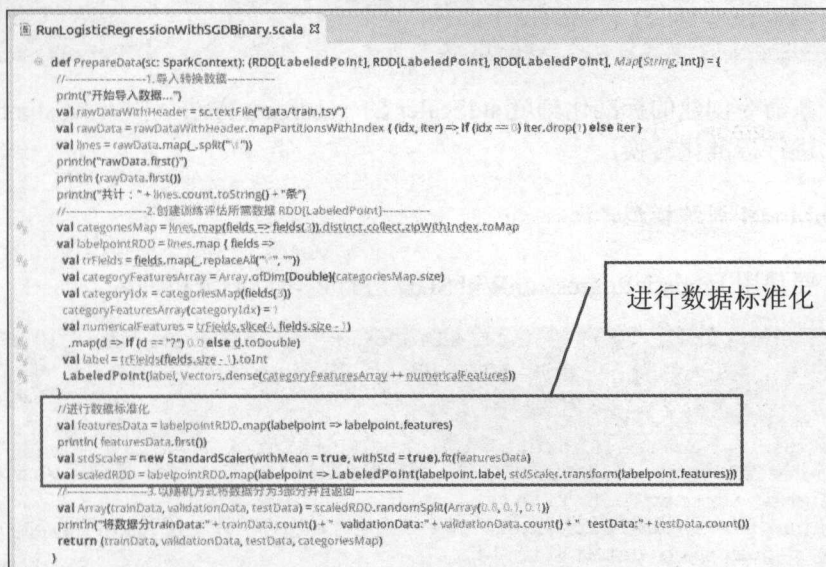


图 14-4 修改 PrepareData() 加入进行数据标准化的程序代码

MLlib 提供了 StandardScaler 链接库帮助进行标准化。首先, 必须在程序代码开头先导入 StandardScaler 链接库。

```
import org.apache.spark.mllib.feature.StandardScaler
```

进行数据标准化程序如下:

```
//进行数据标准化
val featuresData = labelpointRDD.map(labelpoint => labelpoint.features)
val stdScaler =
  new StandardScaler(withMean = true, withStd = true).fit(featuresData)
val scaledRDD = labelpointRDD.map(labelpoint =>
  LabeledPoint(labelpoint.label, stdScaler.transform(labelpoint.features)))
```

上述程序的详细说明如下:

➤ 获取特征字段

```
val featuresData = labelpointRDD.map(labelpoint => labelpoint.features)
```

从 labelpointRDD 使用 map 获取特征字段 featuresData。

➤ 创建标准化刻度

```
val stdScaler =  
  new StandardScaler(withMean = true, withStd = true).fit(featuresData)
```

使用 new StandardScaler 创建标准化刻度 stdScaler，传入 withMean 与 withStd 参数，并且使用 fit 方法传入 featuresData 特征字段。

➤ 进行标准化转换

```
val scaledRDD = labelpointRDD.map(labelpoint =>  
  LabeledPoint(labelpoint.label,  
    stdScaler.transform(labelpoint.features)))
```

使用上一条命令创建的标准化刻度 stdScaler 的 transform 方法，传入 labelpoint.features 特征字段参数，进行标准化转换。

步骤 03 trainModel 训练模型

接下来，要使用 LogisticRegressionWithSGD 进行逻辑回归分析训练。

```
def trainModel(trainData: RDD[LabeledPoint], numIterations: Int,  
  stepSize: Double, miniBatchFraction: Double):  
  (LogisticRegressionModel, Double) = {  
    val startTime = new DateTime()  
    val model = LogisticRegressionWithSGD.train(trainData,  
      numIterations, stepSize, miniBatchFraction)  
    val endTime = new DateTime()  
    val duration = new Duration(startTime, endTime)  
    (model, duration.getMillis())  
  }
```

在 LogisticRegressionWithSGD 使用 Stochastic Gradient Descent（简称 SGD）梯度下降法方式求得最佳解，所以必须输入下列参数：

LogisticRegressionWithSGD.train(trainData, numIterations, stepSize, miniBatchFraction)		
返回	LogisticRegressionModel	
参数	类型	说明
input	RDD[LabeledPoint]	输入的训练数据
numIterations	Int	使用 SGD 迭代次数，默认为 100
stepSize	Double	每次执行 SGD 迭代步长大小，默认为 1
miniBatchFraction	Double	每次迭代参与计算的样本比例，数值为 0~1 之间，默认为 1

步骤 04 parametersTunning 函数

在之前的章节中，我们使用如下命令进行训练，会返回 model 训练完成的模型：

```
LogisticRegressionWithSGD.train(trainData, numIterations,
    stepSize, miniBatchFraction)
```

其中参数 numIterations、stepSize、miniBatchFraction 的设置会影响结果的准确度，以及训练所需的时间。接下来，将调校找出最佳的参数组合。

创建 parametersTunning 参数调校函数如下：

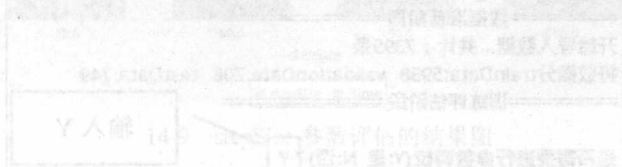
```
def parametersTunning(trainData: RDD[LabeledPoint],
    validationData: RDD[LabeledPoint]): LogisticRegressionModel = {
    println("-----评估 numIterations 参数使用 5, 10, 20,60,100-----")
    evaluateParameter(trainData, validationData, "numIterations",
        Array(5, 15, 20, 60, 100), Array(100), Array(1))
    println("-----评估 stepSize 参数使用 (10,50,100)-----")
    evaluateParameter(trainData, validationData, "stepSize",
        Array(100), Array(10, 50, 100, 200), Array(1))
    println("-----评估 miniBatchFraction 参数使用 (0.5,0.8,1)-----")
    evaluateParameter(trainData, validationData, "miniBatchFraction",
        Array(100), Array(100), Array(0.5, 0.8, 1))
    println("-----所有参数交叉评估找出最好的参数组合-----")
    val bestModel = evaluateAllParameter(trainData, validationData,
        Array(5, 15, 20,60,100),
        Array(10, 50, 100, 200), Array(0.5, 0.8, 1))
    return (bestModel)
}
```

在这里我们评估的参数共有 3 个：numIterations、stepSize、miniBatchFraction。我们希望知道不同的参数值对准确率的影响，以及执行所需要的时间。评估的方法是同一时间只评估一个参数。最后再以 evaluateAllParameter 函数交叉评估 3 种参数，希望找出最好的参数组合。详细请参考第 13 章的说明。

14.3

运行 RunLogisticRegression
WithSGDBinary.scala 进行参数调校

步骤 01 依次选择菜单及其菜单项 Run→Run Configurations (见图 14-5)



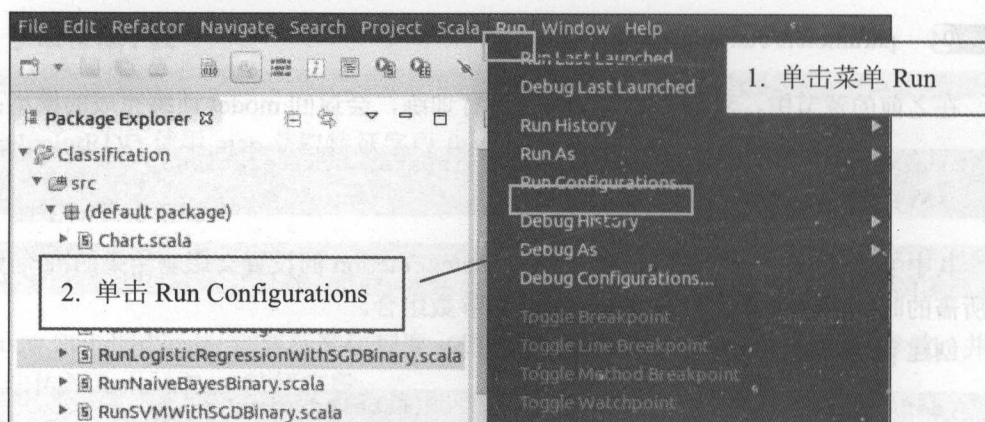


图 14-5 依次选择菜单及其菜单项 Run→Run Configurations

步骤 02 设置 Run Configurations (见图 14-6)

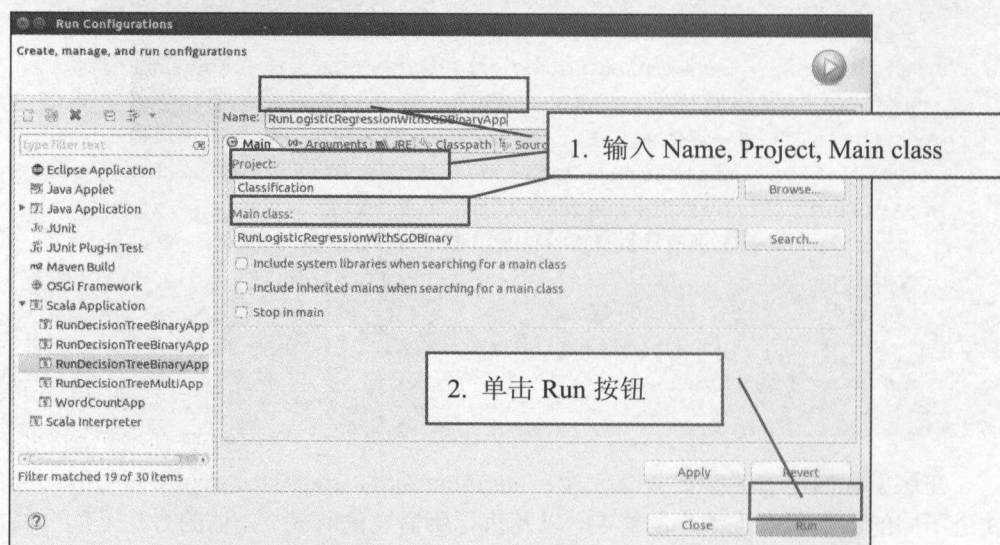
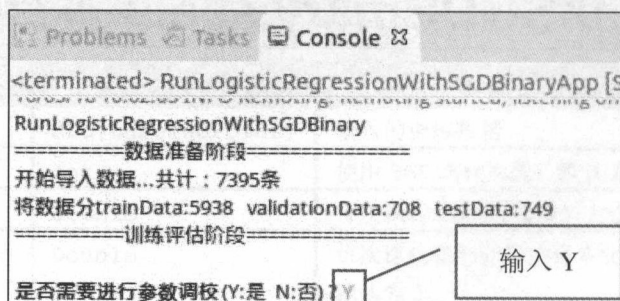


图 14-6 设置 Run Configurations (运行配置)

步骤 03 询问是否进行参数调校

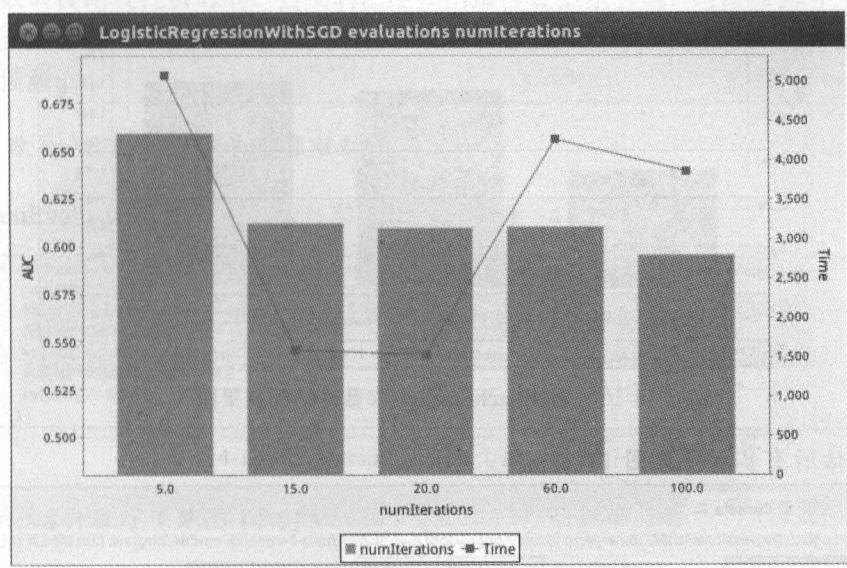
程序会询问是否需要进行参数调校 (Y: 是、N: 否)? 输入 Y。如图 14-7 所示。



14-7 选择进行参数调校

步骤 04 numIterations 参数评估

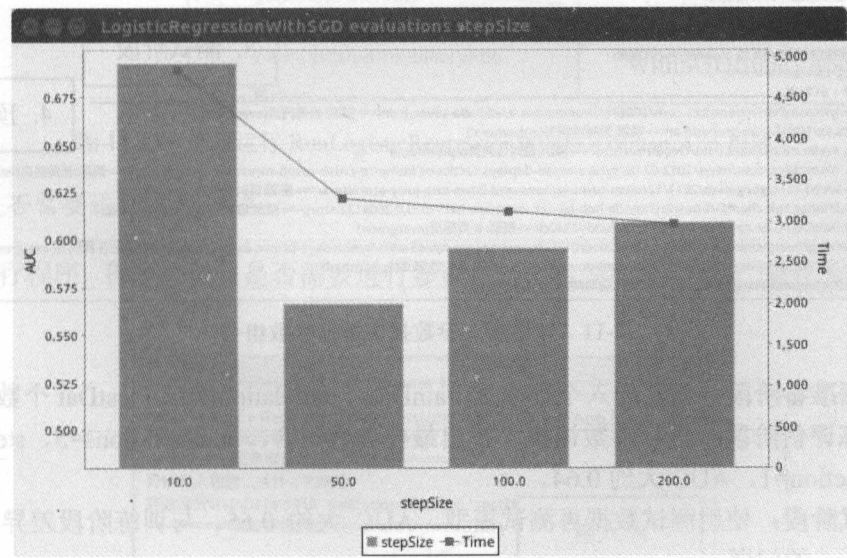
接下来，程序会显示如图 14-8 所示的结果图。numIterations 参数评估：柱形图代表 AUC、折线图代表运行的时间。从图 14-8 可以看到 numIterations=5 时 AUC 最高，而运行的时间 numIterations=5 花最多时间。



14-8 numIterations 参数评估的结果图

步骤 05 stepSize 参数评估

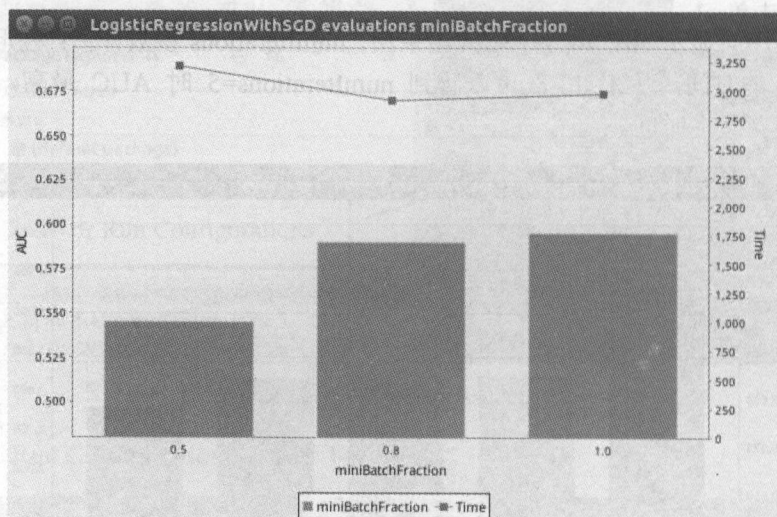
从图 14-9 可以看到，stepSize=10 时，AUC 最高。随着 stepSize 越大所需时间越少。



14-9 stepSize 参数评估的结果图

步骤 06 miniBatchFraction 参数评估

从图 14-10 可以看到, miniBatchFraction=1 时, AUC 最高。所需的时间没有太大的差异。



14-10 miniBatchFraction 参数评估的结果图

步骤 07 评估所有参数 (见图 14-11)

Problems
Tasks
Console

```

<terminated> RunLogisticRegressionWithSGDBinaryApp [Scala Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (2016年5月18日 上午10:09:21)
RunLogisticRegressionWithSGDBinary
=====数据准备阶段=====
开始导入数据...共计: 7395条
将数据分trainData:5905 validationData:739 testData:751
=====训练评估阶段=====

是否需要参数调校(Y:是 N:否)? Y
——评估 numIterations 参数使用 5, 10, 20, 60, 100——
——评估 stepSize 参数使用 (10, 50, 100)——
——评估 miniBatchFraction 参数使用 (0.5, 0.8, 1)——
——所有参数交叉评估找出最好的参数组合——
调校后最佳参数: numIterations:5 ,stepSize:10.0 ,miniBatchFraction:1.0 ,结果AUC=0.6417020964667937

=====测试阶段=====
使用testData测试最佳模型,结果 AUC:0.6565695967318685

=====预测数据=====
开始导入数据...共计: 3171条
网址: http://www.lynnskitchenadventures.com/2009/04/homemade-enchilada-sauce.html==>预测:长青网页(evergreen)
网址: http://lolpics.se/18552-stun-grenade-ar==>预测:暂时性网页(ephemeral)
网址: http://www.xcelerationfitness.com/treadmills.html==>预测:暂时性网页(ephemeral)
网址: http://www.bloomberg.com/news/2012-02-06/syria-s-assad-deploys-tactics-of-father-to-crush-revolt-threatening-reign.html==>预测:长青网页(evergreen)
网址: http://www.wired.com/gadgetlab/2011/12/stem-turns-lemons-and-limes-into-juicy-atomizers/==>预测:暂时性网页(ephemeral)
网址: http://www.latimes.com/health/boostershots/la-heb-fat-tax-denmark-20111013,0,2603132.story==>预测:暂时性网页(ephemeral)
网址: http://www.howlifeworks.com/a/a?AG_ID=1186&cid=7340ci==>预测:长青网页(evergreen)
网址: http://romancingthetoveblog.wordpress.com/2010/01/13/sweet-potato-ravioli-with-lemon-sage-brown-butter-sauce/==>预测:长青网页(evergreen)
网址: http://www.funniez.net/Funny-Pictures/turn-men-down.html==>预测:暂时性网页(ephemeral)
网址: http://youfellasleepwatchingadvd.com/==>预测:暂时性网页(ephemeral)

```

1. 数据准备

2. 训练评估阶段, 进行参数调校找出最佳参数组合

3. 测试阶段

4. 预测数据

14-11 评估所有参数找出最佳参数组合

- 数据准备阶段:** 显示导入个数以及 trainData、validationData、testData 个数。
- 训练评估阶段:** 进行参数调校, 找出最佳参数组合: numIterations=5、stepSize=10、miniBatchFraction=1, AUC 大约 0.64。
- 测试阶段:** 使用测试数据再测试模型, AUC 大约 0.65。与训练阶段差异不大, 确认没有 overfitting 的问题。
- 预测阶段:** 使用 test.tsv 预测是否为长青网页或暂时性网页。预测的数据会显示网址, 可以用浏览器查看。

14.4

运行 RunLogisticRegressionWithSGDBinary.scala 不进行参数调校

因为参数调校比较花费时间。当我们已经找出最佳参数组合时，可以修改 trainEvaluate 程序，改为最佳参数组合。下次要运行预测程序可以不需要再运行参数调校，就能够直接使用最佳参数进行预测。

步骤 01 修改 trainEvaluate 为最佳参数组合

修改 trainEvaluate 程序，改为最佳参数组合，如图 14-12 所示。

```
def trainEvaluate(trainData: RDD[LabeledPoint], validationData: RDD[LabeledPoint]): LogisticRegressionModel = {
    print("开始训练...")
    val (model, time) = trainModel(trainData, 5, 10, 1)
    print("训练完成,所需时间:" + time + "毫秒")
    val AUC = evaluateModel(model, validationData)
    print("评估结果AUC=" + AUC)
    return (model)
}
```

修改为最佳参数组合

14-12 修改 trainEvaluate 为最佳参数组合

步骤 02 开始运行程序（见图 14-13）



图 14-13 开始运行 RunLogisticRegressionWithSGDBinaryApp 程序

步骤 03 是否需要进行参数调校

再次运行程序，程序会询问是否需要参数调校（Y：是、N：否）？输入 N。如图 14-14 所示。

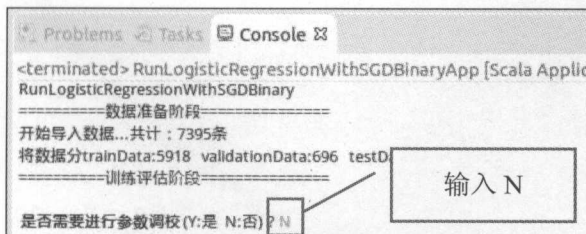


图 14-14 选择不进行参数调校

步骤 04 再次运行程序

运行结果如图 14-15 所示。可以看到程序训练完成后，就可以进行预测。因为不进行参数调校，所以运行所需时间比较少。

```
Problems Tasks Console
<terminated> RunLogisticRegressionWithSGDBinaryApp [Scala Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (2016年5月18日 上午10:
RunLogisticRegressionWithSGDBinary
=====数据准备阶段=====
开始导入数据...共计: 7395条|
将数据分trainData:5918 validationData:696 testData:781
=====训练评估阶段=====

是否需要参数调校(Y:是 N:否) ? N
开始训练...训练完成,所需时间:1970.0毫秒
评估结果AUC=0.6162262592898431
=====测试阶段=====
使用testata测试最佳模型,结果 AUC:0.631818420396541
=====预测数据=====
开始导入数据...共计: 3171条
网址: http://www.lynnskitchenadventures.com/2009/04/homemade-enchilada-sauce.html==>预测: 长青网页(evergreen)
网址: http://lolpics.se/18552-stun-grenade-ar==>预测: 暂时性网页(ephemeral)
网址: http://www.xcelerationfitness.com/treadmills.html==>预测: 暂时性网页(ephemeral)
网址: http://www.bloomberg.com/news/2012-02-06/syria-s-assad-deploys-tactics-of-father-to-crush-revolt-threatening-reign.html==>预测: 长青网页(evergreen)
网址: http://www.wired.com/gadgetlab/2011/12/stem-turns-lemons-and-limes-into-juicy-atomizers/=>预测: 暂时性网页(ephemeral)
网址: http://www.latimes.com/health/boostershots/la-heb-fat-tax-denmark-20111013,0,2603132.story==>预测: 暂时性网页(ephemeral)
网址: http://www.howlifeworks.com/a/a?AG_ID=1186&cid=7340ci==>预测: 暂时性网页(ephemeral)
网址: http://romancingthetoveblog.wordpress.com/2010/01/13/sweet-potato-ravioli-with-lemon-sage-brown-butter-sauce/=>预测: 长青网页(evergreen)
网址: http://www.funniez.net/Funny-Pictures/turn-men-down.html==>预测: 暂时性网页(ephemeral)
网址: http://youfellasleepwatchingadvd.com/=>预测: 暂时性网页(ephemeral)
```

图 14-15 选择最佳参数组合且不进行参数调校的预测程序的运行结果

图15-1 支持向量机SVM二元分类示意图

图15-2 支持向量机SVM二元分类示意图

支持向量机SVM二元分类

图15-3 支持向量机SVM二元分类示意图

图15-4 支持向量机SVM二元分类示意图

图15-5 支持向量机SVM二元分类示意图

图15-6 支持向量机SVM二元分类示意图

图15-7 支持向量机SVM二元分类示意图

第 15 章

支持向量机SVM二元分类

图15-8 支持向量机SVM二元分类示意图

图15-9 支持向量机SVM二元分类示意图

图15-10 支持向量机SVM二元分类示意图

图15-11 支持向量机SVM二元分类示意图

图15-12 支持向量机SVM二元分类示意图

图15-13 支持向量机SVM二元分类示意图

图15-14 支持向量机SVM二元分类示意图

15.1 支持向量机 SVM 算法的基本概念

15.2 RunSVMWithSGDBinary.scala 程序说明

15.3 运行 SVMWithSGD.scala 进行参数调校

15.4 运行 SVMWithSGD.scala 不进行参数调校



图 15-1 支持向量机 SVM 二元分类示意图

图 15-2 支持向量机 SVM 二元分类示意图

图 15-3 支持向量机 SVM 二元分类示意图

图 15-4 支持向量机 SVM 二元分类示意图

图 15-5 支持向量机 SVM 二元分类示意图

图 15-6 支持向量机 SVM 二元分类示意图

图 15-7 支持向量机 SVM 二元分类示意图

图 15-8 支持向量机 SVM 二元分类示意图

图 15-9 支持向量机 SVM 二元分类示意图

图 15-10 支持向量机 SVM 二元分类示意图

图 15-11 支持向量机 SVM 二元分类示意图

图 15-12 支持向量机 SVM 二元分类示意图

图 15-13 支持向量机 SVM 二元分类示意图

图 15-14 支持向量机 SVM 二元分类示意图

图 15-15 支持向量机 SVM 二元分类示意图

图 15-16 支持向量机 SVM 二元分类示意图

图 15-17 支持向量机 SVM 二元分类示意图

图 15-18 支持向量机 SVM 二元分类示意图

图 15-19 支持向量机 SVM 二元分类示意图

本章将使用第 12 章介绍的 StumbleUpon 数据集，运用支持向量机 SVM 二元分类来预测网页是暂时性的或是长青的，并且进行参数调校找出最佳参数组合，提高预测准确度。

15.1 支持向量机 SVM 算法的基本概念

支持向量机(Support Vector Machine)简称为 SVM。SVM 可以用来作为分类(Classification)的工具。在说明 SVM 分类的方法之前，我们先看下图（见图 15-1）。图中有圆点以及三角形的点，希望找出一条线段进行比较好的分类。

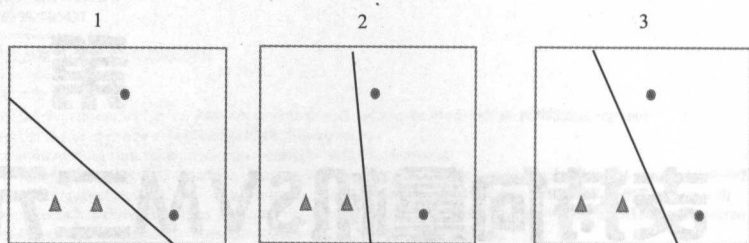


图 15-1 不同分类的示意图

SVM 主要是在寻找最大边界的超平面（**Maximum Marginal Hyperplane**），因为其具有较高的分类准确性，并且有较高的误差容忍度。如图 15-2 所示的虚线范围，其中的第 3 张图分类具有最大的边际区，所以对于 SVM 算法而言，此图是比较好的分类。

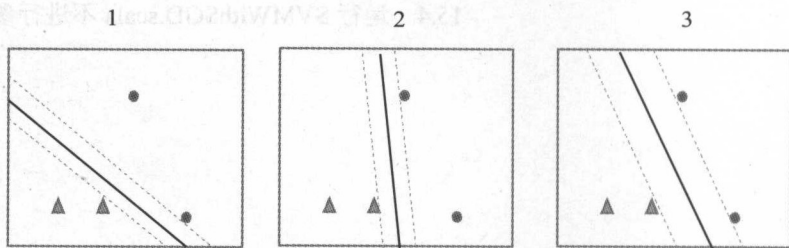


图 15-2 找到比较好的分类的示意图

15.2 RunSVMWithSGDBinary.scala 程序说明

我们已经使用 Spark MLlib 创建了支持向量机分析 RunSVMWithSGDBinary.scala，完整的程序代码可参考本书范例程序。程序的架构与之前第 13 章介绍的 RunDecisionTreeBinary.scala 类似，可以参考第 12 章的详细说明。以下仅说明 RunSVMWithSGDBinary.scala 重要的修改部分。

步骤 01 导入 SVMWithSGD 链接库（见图 15-3）


```
*RunSVMWithSGDBinary.scala ❏
import org.apache.log4j.Logger
import org.apache.log4j.Level
import org.apache.spark.storage.StorageLevel
import org.apache.spark.rdd.RDD
import org.apache.spark.{SparkConf, SparkContext}
import org.apache.spark.mllib.regression.LabeledPoint
import org.apache.spark.mllib.evaluation._
import org.apache.spark.mllib.linalg.Vectors
import org.joda.time._
import org.jfree.data.category.DefaultCategoryDataset
import org.apache.spark.mllib.feature.StandardScaler
import org.apache.spark.mllib.classification.SVMWithSGD
import org.apache.spark.mllib.classification.SVMModel
```

import 相关链接库

图 15-3 导入 SVMWithSGD 链接库

首先必须导入 Support Vector Machine 相关链接库

```
import org.apache.spark.mllib.classification.SVMWithSGD
import org.apache.spark.mllib.classification.SVMModel
```

步骤 02 修改 PrepareData() 以便加入数据标准化的程序代码

我们将把第 12 章 StumbleUpon 数据集运用于支持向量机 SVM 二元分类。

下列程序（见图 15-4）基本上与第 13 章决策树数据准备 PrepareData() 类似。唯一不同的是，当我们进行回归分析算法时，必须将数值特征（Numerical Features）字段进行标准化。因为数值特征字段单位不同而数字差异很大，所以彼此无法比较。这时就要使用标准化，让数值特征字段有共同的标准。

```
RunSVMWithSGDBinary.scala ❏

def PrepareData(sc: SparkContext): (RDD[LabeledPoint], RDD[LabeledPoint], RDD[LabeledPoint], Map[String, Int]) = {
  //-----1. 导入转换数据-----
  print("开始导入数据...")
  val rawDataWithHeader = sc.textFile("data/train.tsv")
  val rawData = rawDataWithHeader.mapPartitionsWithIndex { (idx, iter) => if (idx == 0) iter.drop(1) else iter }
  val lines = rawData.map(_._1).split("\t")
  println("共计: " + lines.count().toString() + "条")
  //-----2. 创建训练评估所需数据 RDD[LabeledPoint]-----
  val categoriesMap = lines.map(fields => fields(3)).distinct.collect.zipWithIndex.toMap
  val labelpointRDD = lines.map { fields =>
    val rFields = fields.map(_._1).replaceAll("[", "")
    val categoryFeaturesArray = Array.ofDim[Double](categoriesMap.size)
    val categoryIdx = categoriesMap(fields(3))
    categoryFeaturesArray(categoryIdx) = 1
    val numericalFeatures = rFields.slice(4, fields.size - 1)
    .map(d => if (d == "?") 0.0 else d.toDouble)
    val label = rFields(fields.size - 1).toInt
    LabeledPoint(label, Vectors.dense(categoryFeaturesArray ++ numericalFeatures))
  }
  //进行数据标准化
  val featuresData = labelpointRDD.map(labelpoint => labelpoint.features)
  val stdScaler = new StandardScaler(withMean = true, withStd = true).fit(featuresData)
  val scaledRDD = labelpointRDD.map(labelpoint => LabeledPoint(labelpoint.label, stdScaler.transform(labelpoint.features)))
  //-----3. 以随机方式将数据分为3部分并且返回
  val Array(trainData, validationData, testData) = scaledRDD.randomSplit(Array(0.8, 0.1, 0.1))
  println("将数据分为trainData:" + trainData.count() + " validationData:" + validationData.count() + " testData:" + testData.count())
  return (trainData, validationData, testData, categoriesMap) //返回数据
}
```

加入数据标准化的程序代码

图 15-4 修改 PrepareData() 以便加入数据标准化的程序代码

步骤 03 trainModel 训练模型

接下来要使用 SVMWithSGD 进行逻辑回归分析训练。

```
def trainModel(trainData: RDD[LabeledPoint], numIterations: Int, stepSize: Double, regParam: Double): (SVMModel, Double) = {  
    val startTime = new DateTime()  
    val model = SVMWithSGD.train(trainData, numIterations, stepSize, regParam)  
    val endTime = new DateTime()  
    val duration = new Duration(startTime, endTime)  
    (model, duration.getMillis())  
}
```

在 SVMWithSGD 使用 Stochastic Gradient Descent（简称 SGD）随机梯度下降法方式求得最佳解，所以必须输入下列参数：

SVMWithSGD.train(trainData, numIterations, stepSize, miniBatchFraction)		
返回	LogisticRegressionModel	
参数	类型	说明
input	RDD[LabeledPoint]	输入的训练数据
numIterations	Int	使用 SGD 迭代次数，默认为 100
stepSize	Double	每次执行 SGD 迭代步长的大小，默认为 1
regParam	Double	正则化参数，数值为 0~1 之间

步骤 04 parametersTunning 函数

输入 parametersTunning 函数如下：

```
def parametersTunning(trainData: RDD[LabeledPoint], validationData: RDD[LabeledPoint]): SVMModel = {  
    println("-----评估 numIterations 参数使用 1, 3, 5, 15, 25-----")  
    evaluateParameter(trainData, validationData, "numIterations", Array(1, 3, 5, 15, 25), Array(100), Array(1))  
    println("-----评估 stepSize 参数使用 (10, 50, 100, 200)-----")  
    evaluateParameter(trainData, validationData, "stepSize", Array(25), Array(10, 50, 100, 200), Array(1))  
    println("-----评估 regParam 参数使用 (0.01, 0.1, 1)-----")  
    evaluateParameter(trainData, validationData, "regParam", Array(25), Array(100), Array(0.01, 0.1, 1))  
    println("-----所有参数交叉评估找出最好的参数组合-----")  
    val bestModel = evaluateAllParameter(trainData, validationData, Array(1, 3, 5, 15, 25), Array(10, 50, 100, 200), Array(0.01, 0.1, 1))  
    return (bestModel)  
}
```

在这里评估的参数共有 3 个：numIterations、stepSize、regParam。我们希望得知不同的参数值对准确率的影响，以及运行所需要的时间。评估的方法是同时只评估一个参数。最后以 evaluateAllParameter 函数交叉评估 3 种参数，希望找出最好的参数组合。

15.3 运行 SVMWithSGD.scala 进行参数调校

步骤 01 依次选择菜单及其菜单项 Run→Run Configurations（见图 15-5）

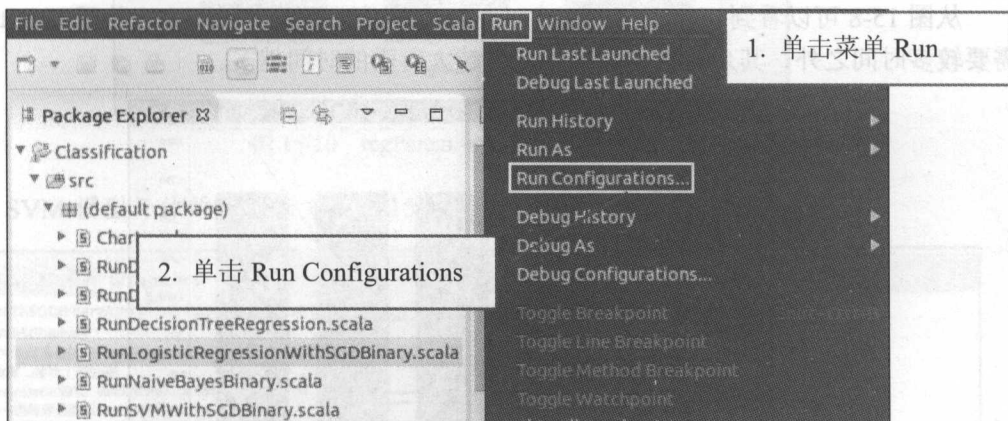


图 15-5 依次选择菜单及其菜单项 Run→Run Configurations

步骤 02 设置 Run Configurations（见图 15-6）

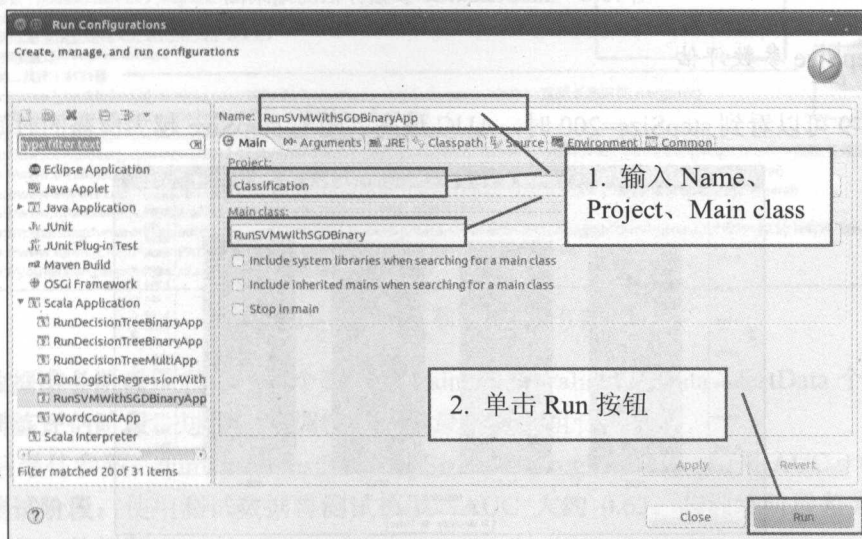


图 15-6 设置 Run Configurations（运行配置）

步骤 03 询问是否进行参数调校

程序会询问是否需要进行参数调校（Y：是、N：否）？输入 Y。如图 15-7 所示。

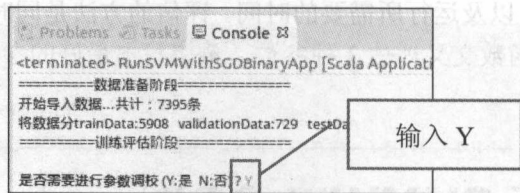


图 15-7 选择进行参数调校

步骤 04 numIterations 参数评估

从图 15-8 可以看到 numIterations=1 时，AUC 最高，不过差异不大。除了 numIterations=1 需要较多时间之外，其余随着 numIterations 越大所需的时间越多。

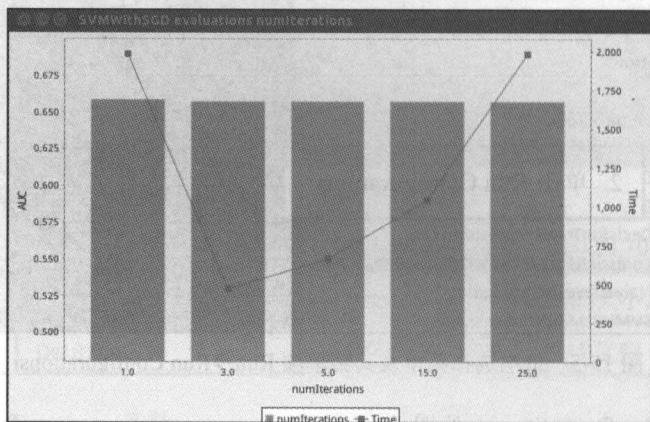


图 15-8 numIterations 参数评估的结果图

步骤 05 stepSize 参数评估

从图 15-9 可以看到 stepSize=200 时，AUC 最高。随着 stepSize 越大所需的时间越少。

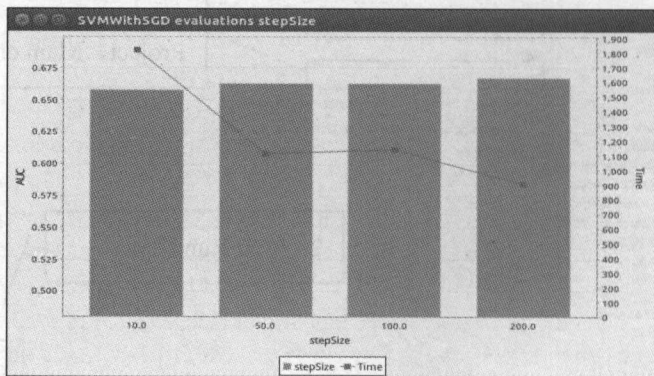


图 15-9 stepSize 参数评估的结果图

步骤 06 regParam 参数评估

从图 15-10 可以看到 $\text{regParam}=1$ 时, AUC 最高。随着 regParam 越大所需时间越少。

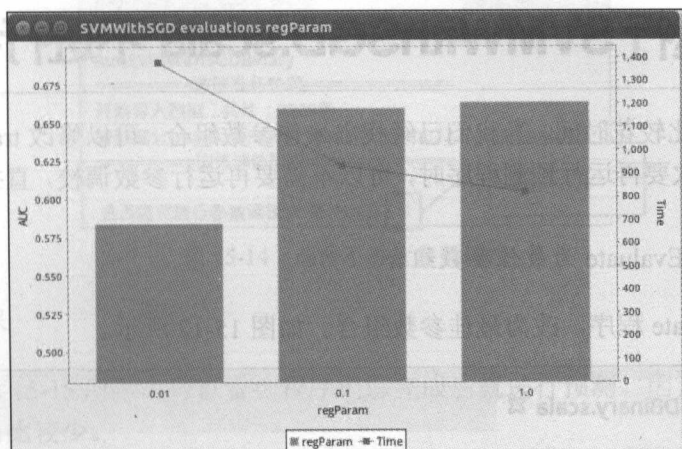


图 15-10 regParam 参数评估的结果图

步骤 07 SVM 模型评估所有参数 (见图 15-11)

```

RunSVMWithSGDBinaryApp [Scala Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (2016年5月18日 下午1:29:16)
RunSVMWithSGDBinary
=====数据准备阶段=====
开始导入数据...共计: 7395条
将数据分trainData:5910 validationData:751 testData:734
=====训练评估阶段=====

是否需要参数调校(Y:是 N:否)? Y
|-----评估 numIterations参数使用 1, 3, 5, 15, 25-----
|-----评估 stepSize参数使用 (10, 50, 100, 200)-----
|-----评估 regParam参数使用 (0.01, 0.1, 1)-----
|-----所有参数交叉评估找出最好的参数组合-----
调校后最佳参数: numIterations:25 ,stepSize:100.0 ,regParam:1.0 ,结果AUC = 0.6540131074985106
-----测试阶段-----
使用testata测试最佳模型,结果AUC:0.637036621404257
-----预测数据-----
开始导入数据...共计: 3171条
网址: http://www.lynnskitchenadventures.com/2009/04/homemade-enchilada-sauce.html==>预测:长青网页(evergreen)
网址: http://lolpics.se/18552-stun-grenade-ar==>预测:暂时性网页(ephemeral)
网址: http://www.xcelerationfitness.com/treadmills.html==>预测:长青网页(evergreen)
网址: http://www.bloomberg.com/news/2012-02-06/syria-s-assad-deploys-tactics-of-father-to-crush-revolt-threatening-reign.html==>预测:长青网页(evergreen)
网址: http://www.wired.com/gadgetlab/2011/12/stem-turns-lemons-and-limes-into-juicy-atomizers/==>预测:暂时性网页(ephemeral)
网址: http://www.latimes.com/health/boostershots/la-heb-fat-tax-denmark-20111013,0,2603132.story==>预测:暂时性网页(ephemeral)
网址: http://www.howlfeetworks.com/a/a7AG_ID=1186&cid=7340ci==>预测:暂时性网页(ephemeral)
网址: http://romancingthetovestoveblog.wordpress.com/2010/01/13/sweet-potato-ravioli-with-lemon-sage-brown-butter-sauce/==>预测:长青网页(evergreen)
网址: http://www.funniez.net/Funny-Pictures/turn-men-down.html==>预测:暂时性网页(ephemeral)
网址: http://youfelasleepwatchingadvd.com/==>预测:暂时性网页(ephemeral)
  
```

图 15-11 SVM 模型评估所有参数

1. **数据准备阶段:** 显示导入个数以及 trainData、validationData、testData 个数。

2. **训练评估阶段:** 进行参数调校, 找出最佳参数组合。

调校后最佳参数: numIterations:25、stepSize:100、regParam:1, 结果 $\text{AUC} = 0.65$ 。

3. **测试阶段:** 使用测试数据再测试模型, AUC 大约 0.63。与训练阶段差异不大, 确认没有 overfitting 的问题。

4. **预测阶段:** 使用 test.tsv 预测是否为长青网页或暂时性网页。预测的数据会显示出网址, 可以用浏览器查看。

15.4 运行 SVMWithSGD.scala 不进行参数调校

因为参数调校比较花时间。当我们已经找出最佳参数组合，可以修改 trainEvaluate 程序为最佳参数组合。下次要再运行预测程序时，可以不需要再运行参数调校，直接使用最佳参数进行预测即可。

步骤 01 修改 trainEvaluate 为最佳参数组合

修改 trainEvaluate 程序，改为最佳参数组合。如图 15-12 所示。

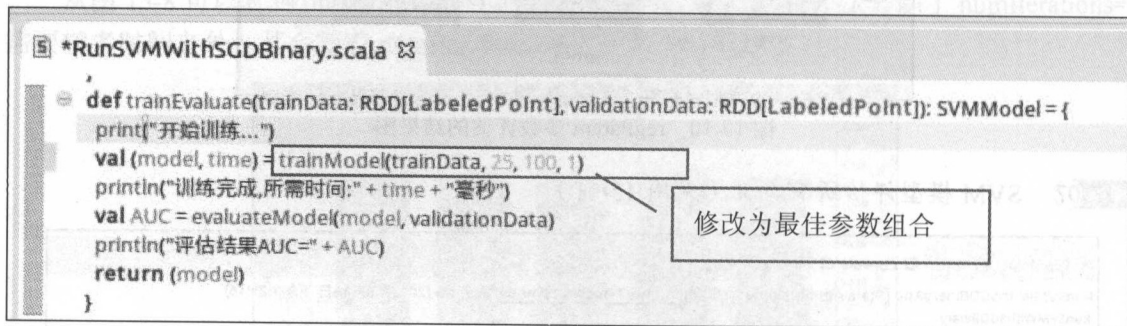


图 15-12 修改 trainEvaluate 为最佳参数组合

步骤 02 开始运行程序（见图 15-13）

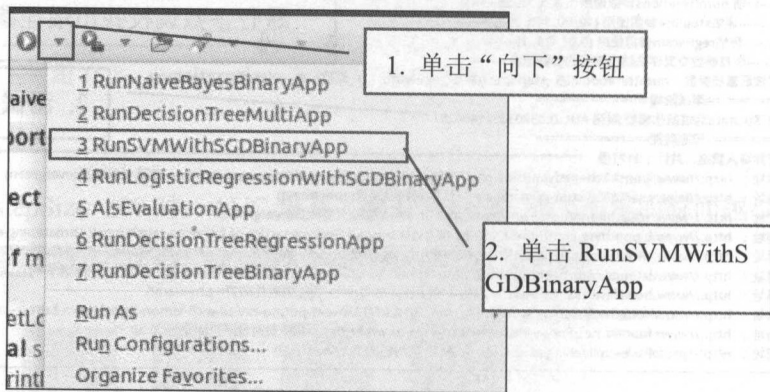


图 15-13 开始运行 RunSVMWithSGDBinaryApp 程序

步骤 03 是否需要进行参数调校

再次运行程序，程序会询问是否需要进行参数调校（Y：是、N：否）？输入 N。如图 15-14 所示。

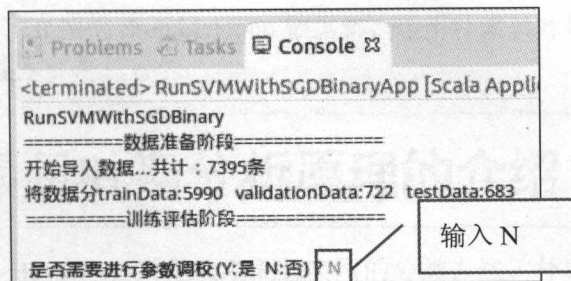


图 15-14 选择不进行参数调校

步骤 04 运行结果

运行结果如图 15-15 所示，可以看到程序训练完成后就进行预测。因为不进行参数调校，所以运行所需时间比较少。

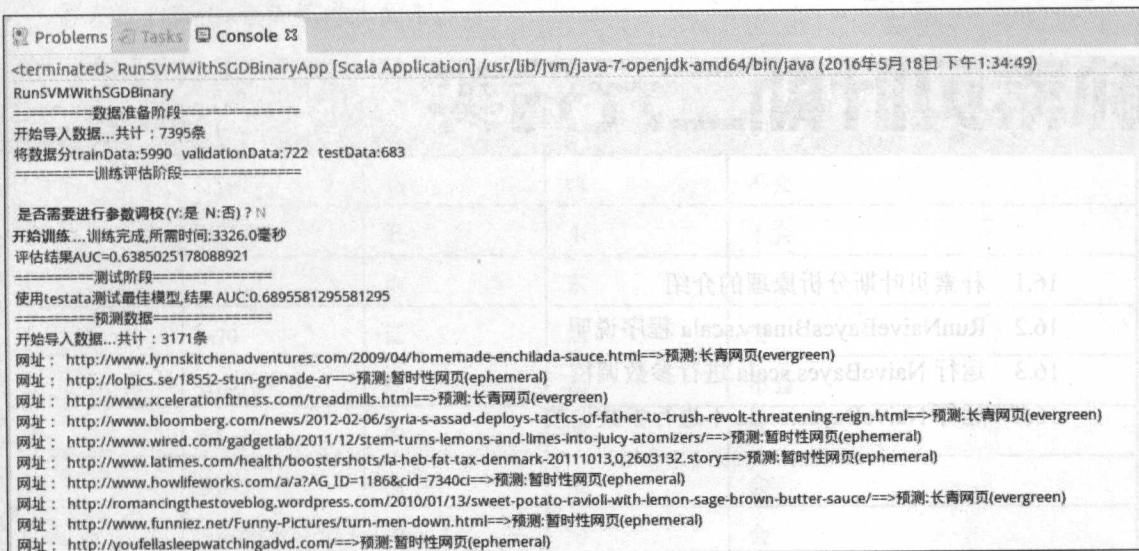


图 15-15 选择最佳参数组合且不进行参数调校的预测程序的运行结果

运行 NaiveBayes.scala 不进行参数调校

因为参数调校比较耗时，所以只调校一次，得到最佳参数组合。下次使用时再输入，所以不需要再进行参数调校，直接使用最佳参数进行预测即可。

❷❷❷❷ 修改 trainEvaluate 为最佳参数组合，如图 15-11 所示。

修改 trainEvaluate 程序，改为最佳参数组合，如图 15-12 所示。

果统计 40

第 16 章

朴素贝叶斯二元分类

- 16.1 朴素贝叶斯分析原理的介绍
- 16.2 RunNaiveBayesBinary.scala 程序说明
- 16.3 运行 NaiveBayes.scala 进行参数调校
- 16.4 运行 NaiveBayes.scala 不进行参数调校

❷❷❷❷ 是否需要再进行参数调校

再次运行程序，程序会提示是否需要再进行参数调校（Y 或 N，否），输入 N，如图 15-14 所示。

本章中将使用第 12 章介绍的 StumbleUpon 数据集，运用朴素贝叶斯（Naïve-Bayes）二元分类来预测网页是暂时性的或是长青的。

16.1 朴素贝叶斯分析原理的介绍

朴素贝叶斯（Naïve-Bayes）分析是简单而且实用的分类方法。朴素贝叶斯分类法是以贝氏定理（Bayes' theorem）为基础。贝氏定理来自于 18 世纪数学家托马斯·贝叶斯。朴素贝叶斯分类法希望能通过概率统计的分析，用以判断未知类的数据应该属于哪一个类。

步骤 01 朴素贝叶斯分析实例

朴素贝叶斯分析通过分析数据中的特征与标签之间的概率，作为分类的依据。

假设我们有 10 条训练样本如下：

项目（特征）	湿度（特征）	气压（特征）	风向（特征）	是否会下雨？ （标签）
1	<50	高	西	不会
2	51~60	低	东	不会
3	51~60	高	东	不会
4	>70	低	西	会
5	61~70	高	西	不会
6	61~70	中	西	会
7	51~60	高	南	会
8	51~60	中	南	会
9	61~70	低	南	会
10	<50	高	北	会

我们想要判断新进样本“高气压、湿度 51~60、西风，是否会下雨？”的时候。

首先，计算“高气压、湿度 51~60、西风，会下雨的概率”；

接下来，计算“高气压、湿度 51~60、西风，不会下雨的概率”；

如果“会下雨的概率”>“不会下雨的概率”，朴素贝叶斯分类预测新进样本“会下雨”。

如果“会下雨的概率”<“不会下雨的概率”，朴素贝叶斯分类预测新进样本“不会下雨”。

步骤 02 计算“高气压、湿度 51~60、西风，会下雨的概率”

首先，计算“高气压、湿度 51~60、西风，会下雨的概率”，方式如下：

$$P(\text{会}) * P(\text{高} | \text{会}) * P(51 \sim 60 | \text{会}) * P(\text{西} | \text{会}) = (6/10) * (2/6) * (2/6) * (2/6) = 0.02222$$

说明如下:

概率条件	计算方式	计算结果
P(会)	(会下雨的个数) / (全部个数)	(6/10)
P(高 会)	(高气压 且 会下雨的个数) / (会下雨的个数)	(2/6)
P(51~60 会)	(湿度 51~60 且 会下雨的个数) / (会下雨的个数)	(2/6)
P(西 会)	(西风 且 会下雨的个数) / (会下雨的个数)	(2/6)
P(会) * P(高 会) * P(51~60 会) *P(西 会)	(6/10)* (2/6) * (2/6) * (2/6)	0.02222

步骤 03 计算“高气压、湿度 51~60、西风，不会下雨的概率”

接下来，计算“高气压、湿度 51~60、西风，不会下雨的概率”，方式如下：
 $P(\text{不会}) * P(\text{高}|\text{不会}) * P(\text{51~60}|\text{不会}) * (\text{西}|\text{不会}) = (4/10) * (3/4) * (2/4) * (2/4) = 0.075$

说明如下：

概率条件	计算方式	计算结果
P(不会)	(不会下雨的个数) / (全部个数)	(4/10)
P(高 不会)	(高气压 且 不会下雨的个数) / (不会下雨的个数)	(3/4)
P(51~60 不会)	(湿度 51~60 且 不会下雨的个数) / (不会下雨的个数)	(2/4)
P(西 不会)	(西风 且 不会下雨的个数) / (不会下雨的个数)	(2/4)
P(不会) * P(高 不会)* P(51~60 不会)* P(西 不会)	(4/10)*(3/4)*(2/4) *(2/4)	0.075

结论：“会下雨”的概率为 0.02222，小于“不会下雨”的概率 0.075，所以朴素贝叶斯分类预测新进样本“不会下雨”。

16.2 RunNaiveBayesBinary.scala 程序说明

我们已经使用 Spark MLlib 创建逻辑回归分析 RunNaiveBayesBinary.scala，完整的程序代码可参考本书提供的范例程序。RunNaiveBayesBinary.scala 程序的架构，与我们之前第 12 章介绍的 RunDecisionTreeBinary.scala 类似，读者可以参考第 12 章的详细说明。以下仅说明 RunSVMWithSGDBinary.scala 重要的修改部分。

步骤 01 导入 SVMWithSGD 链接库（见图 16-1）

```
RunNaiveBayesBinary.scala
import org.apache.log4j.Logger
import org.apache.log4j.Level
import org.apache.spark.storage.StorageLevel
import org.apache.spark.rdd.RDD
import org.apache.spark.{ SparkConf, SparkContext }
import org.apache.spark.mllib.regression.LabeledPoint
import org.apache.spark.mllib.evaluation._
import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.mllib.feature.StandardScaler

import org.joda.time._
import org.jfree.data.category.DefaultCategoryDataset

import org.apache.spark.mllib.classification.NaiveBayes
import org.apache.spark.mllib.classification.NaiveBayesModel
```

import 相关链接库

图 16-1 导入 SVMWithSGD 链接库

首先必须导入朴素贝叶斯分类法的相关链接库。

```
import org.apache.spark.mllib.classification.NaiveBayes
import org.apache.spark.mllib.classification.NaiveBayesModel
```

步骤 02 修改 PrepareData() 以便加入数据标准化的程序代码

我们将第 12.2 节的 StumbleUpon 数据集运用于朴素贝叶斯二元分类。当进行贝氏二元分类时，必须将数值特征字段进行标准化。因为数值特征字段单位不同而数字差异很大，所以彼此无法比较。这时就要使用标准化，让数值特征字段有共同的标准。

修改 PrepareData() 时的屏幕显示界面如图 16-2 所示。

```
*RunNaiveBayesBinary.scala
def PrepareData(sc: SparkContext): (RDD[LabeledPoint], RDD[LabeledPoint], RDD[LabeledPoint], Map[String, Int]) = {
  //-----1. 导入转换数据-----
  print("开始导入数据...")
  val rawDataWithHeader = sc.textFile("data/train.tsv")
  val rawData = rawDataWithHeader.mapPartitionsWithIndex { (idx, iter) => if (idx == 0) iter.drop(1) else iter }
  val lines = rawData.map(_._split("\t"))
  println("共计: " + lines.count().toString() + "条")
  //-----2. 创建训练评估所需数据 RDD[LabeledPoint]-----
  val categoriesMap = lines.map(fields => fields(3)).distinct.collect.zipWithIndex.toMap
  val labelpointRDD = lines.map { fields =>
    val trFields = fields.map(_._replaceAll("\t", ""))
    val categoryFeaturesArray = Array.ofDim[Double](categoriesMap.get(fields(3)).size)
    val categoryIdx = categoriesMap.get(fields(3))
    categoryFeaturesArray(categoryIdx) = 1
    val numericalFeatures = trFields.slice(4, fields.size - 1)
    val d => if (d == "?" || d == "0" || d == "1") d.toDouble else d
    val d => if (d < 0) 0.0 else d
    val label = trFields(fields.size - 1).toInt
    LabeledPoint(label, Vectors.dense(categoryFeaturesArray ++ numericalFeatures))
  }
  val featuresData = labelpointRDD.map(labelpoint => labelpoint.features)
  val stdScaler = new StandardScaler(withMean = false, withStd = true).fit(featuresData)
  val scaledRDD = labelpointRDD.map(labelpoint => LabeledPoint(labelpoint.label, stdScaler.transform(labelpoint.features)))
  //-----3. 以随机方式将数据分为3部分并且返回-----
  val Array(trainData, validationData, testData) = scaledRDD.randomSplit(Array(0.8, 0.1, 0.1))
  println("将数据分trainData: " + trainData.count() + " validationData: " + validationData.count() + " testData: " + testData.count())
  return (trainData, validationData, testData, categoriesMap) //返回数据
}
```

1. NaiveBayes 数值特征字段一定要大于 0，所以负数转换为 0

2. 加入数据标准化 withMean=false

图 16-2 修改 PrepareData() 以便加入数据标准化的程序代码

上述程序与之前第 15.2 节的做法类似，自行参考第 15.2 小节的说明。但是有下列 2 个地方不同：

➤ 数值特征字段一定要大于 0

NaiveBayes 数值特征字段一定要大于 0，所以加入下述命令将负数转换为 0：

```
.map(d => if (d < 0) 0.0 else d)
```

➤ 进行数据标准化，参数 withMean=false

```
val scaler = new StandardScaler(withMean = false, withStd = true).fit(vectors)
```

朴素贝叶斯分类法算法在进行数据标准化时，参数 withMean 必须设置为 false。

步骤 03 trainModel 训练模型

接下来要使用 RunNaiveBayesBinary 进行朴素贝叶斯分类法训练。

```
def trainModel(trainData: RDD[LabeledPoint], lambda: Int):  
  (NaiveBayesModel, Double) = {  
    val startTime = new DateTime()  
    val model = NaiveBayes.train(trainData, lambda)  
    val endTime = new DateTime()  
    val duration = new Duration(startTime, endTime)  
    (model, duration.getMillis())  
  }
```

在 NaiveBayes 必须输入下列参数：

val NaiveBayesModel = NaiveBayes.train(input, lambda)

返回NaiveBayesModel

参数	类型	说明
input	RDD[LabeledPoint]	输入的训练数据
lambda	Int	设置 lambda 参数，默认值：1.0

步骤 04 parametersTunning 函数

输入 parametersTunning 函数如下：

```
def parametersTunning(trainData: RDD[LabeledPoint],  
  validationData: RDD[LabeledPoint]): NaiveBayesModel = {  
  println("-----评估 lambda 参数使用 1, 3, 5, 15, 25-----")  
  evaluateParameter(trainData, validationData, "lambda",  
    Array(1, 3, 5, 15, 25))  
  println("-----所有参数交叉评估找出最好的参数组合-----")  
  val bestModel = evaluateAllParameter(trainData, validationData,
```



```

Array(1, 3, 5, 15, 25))
return (bestModel)
}

```

在这里评估的参数是 `lambda`。我们希望得知不同的参数值对准确率的影响，以及运行所需要的时间。

16.3 运行 NaiveBayes.scala 进行参数调校

步骤 01 依次选择菜单及其菜单选项 `Run`→`Run Configurations` (见图 16-3)

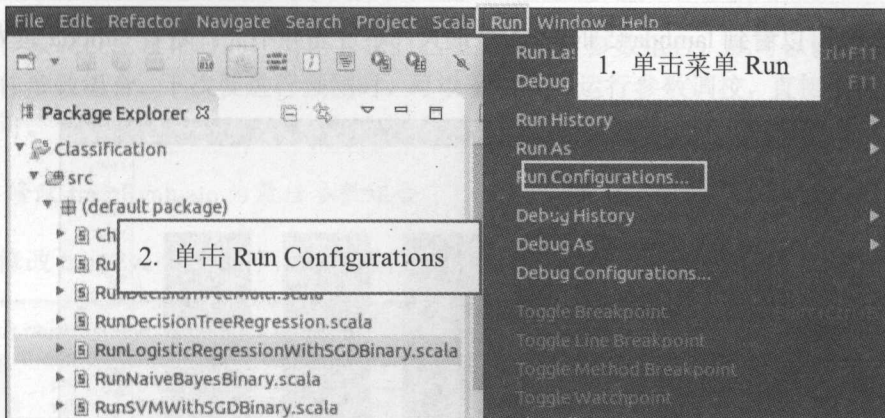


图 16-3 依次选择菜单及其菜单选项 `Run`→`Run Configurations`

步骤 02 设置 `Run Configurations` (见图 16-4)

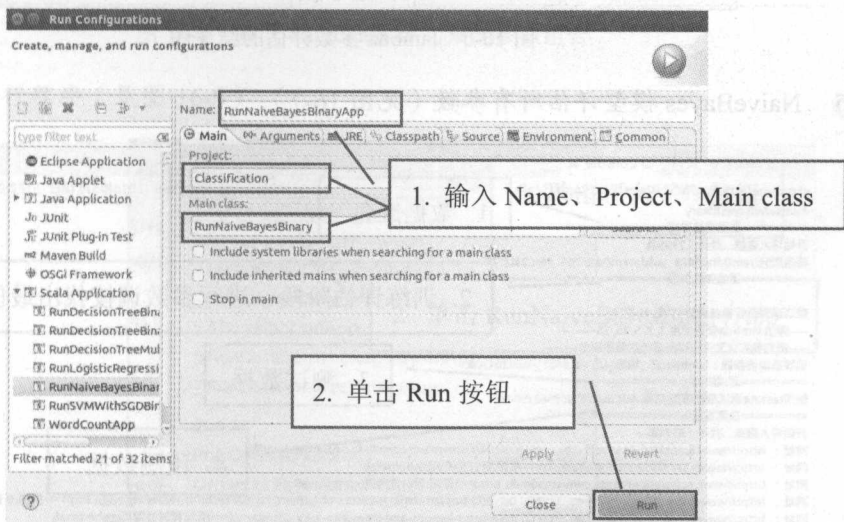


图 16-4 设置 `Run Configurations` (运行配置)

步骤 03 询问是否进行参数调校

程序会询问是否需要进行参数调校 (Y: 是、N: 否)? 输入 Y。如图 16-5 所示。

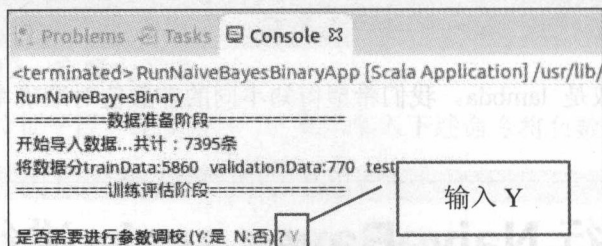


图 16-5 选择进行参数调校

步骤 04 lambda 参数评估

从图 16-6 可以看到 $\lambda=5$ 时, AUC 最大, 不过差异不大。随着 λ 越大所需的时间越少。

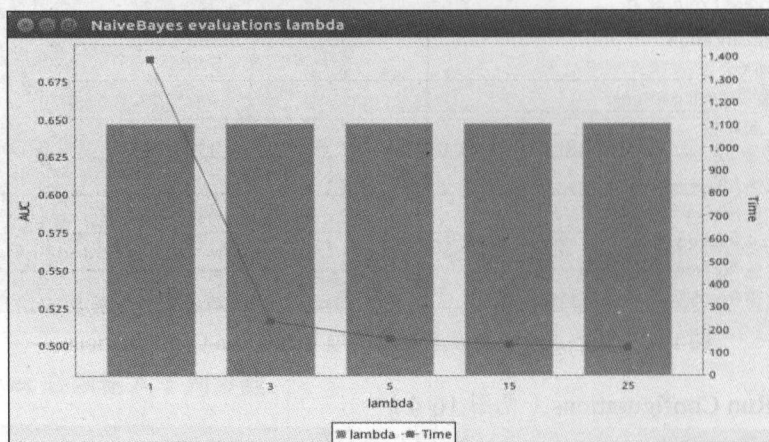


图 16-6 lambda 参数评估的结果图

步骤 05 NaiveBayes 模型评估所有参数 (见图 16-7)

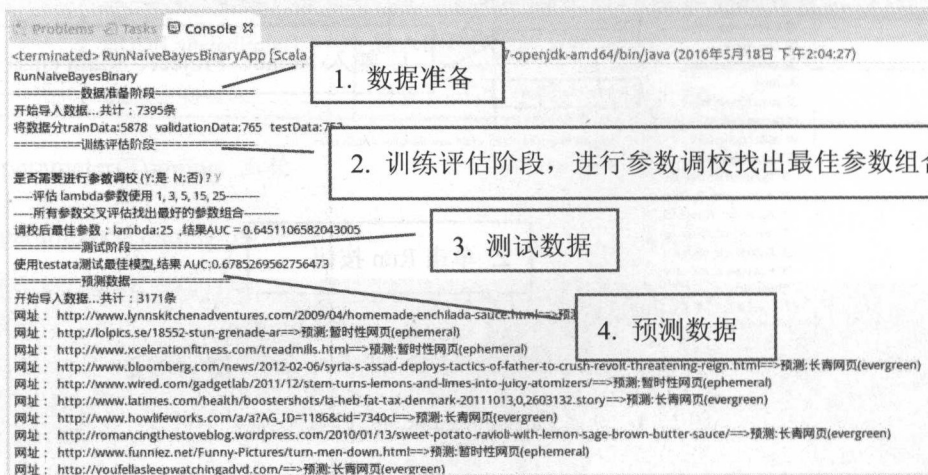


图 16-7 NaiveBayes 模型评估所有参数

- 1. **数据准备阶段**: 显示导入个数以及 trainData、validationData、testData 个数。
- 2. **训练评估阶段**: 进行参数调校, 找出最佳参数组合。调校后最佳参数: lambda:25, 结果 AUC = 0.64。
- 3. **测试阶段**: 使用测试数据再测试模型, AUC 大约 0.67。与训练阶段差异不大, 确认没有 overfitting 的问题。
- 4. **预测阶段**: 使用 test.tsv 预测是否为长青网页或暂时性网页。预测的数据会显示出网址, 可以用浏览器查看。

16.4 运行 NaiveBayes.scala 不进行参数调校

因为参数调校比较花时间, 所以当我们已经找出最佳参数组合, 就可以修改 trainEvaluate 程序为最佳参数组合。下次要进行预测时, 可以不需要再运行参数调校, 直接使用最佳参数进行预测即可。

步骤 01 修改 trainEvaluate 为最佳参数组合

可以修改 trainEvaluate 程序, 改为最佳参数组合, 如图 16-8 所示。

```
*RunNaiveBayesBinary.scala
def trainEvaluate(trainData: RDD[LabeledPoint], validationData: RDD[LabeledPoint]): NaiveBayesModel = {
  print("开始训练...")
  val (model, time) = trainModel(trainData, 25)
  print("训练完成,所需时间:" + time + "毫秒")
  val AUC = evaluateModel(model, validationData)
  print("评估结果AUC=" + AUC)
  return (model)
}
```

图 16-8 修改 trainEvaluate 为最佳参数组合

步骤 02 开始运行程序 (见图 16-9)



图 16-9 开始运行 RunNaiveBayesBinaryApp 程序

步骤 03 是否需要进行参数调校

再次运行程序，程序会询问是否需要进行参数调校（Y：是、N：否）？输入 N。如图 16-10 所示。



图 16-10 选择不进行参数调校

步骤 04 运行结果

运行结果如图 16-11 所示。可以看到程序训练完成后就可以进行预测。因为不进行参数调校，所以运行所需的时间比较少。

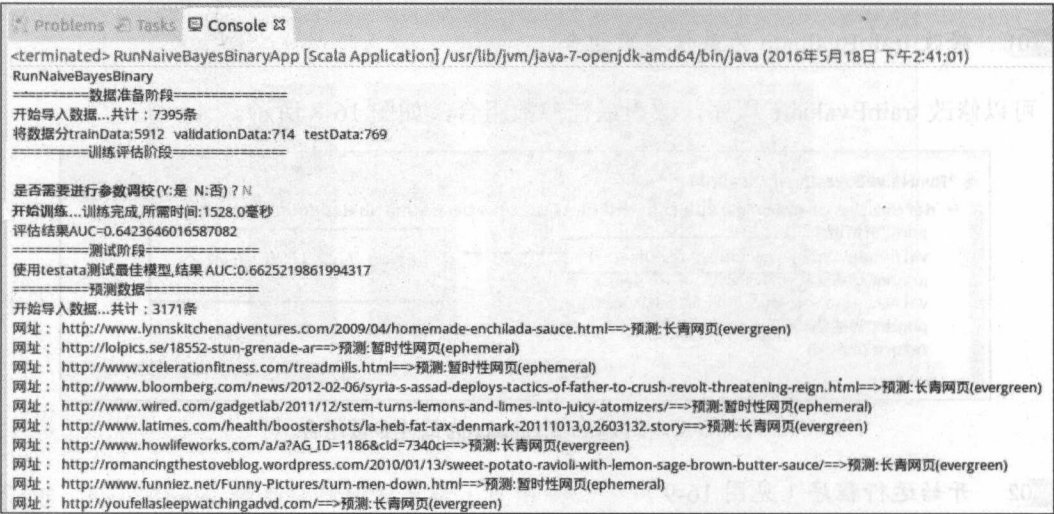


图 16-11 选择最佳参数组合且不再进行参数调校之后的运行结果

本书为《机器学习》系列丛书之一，主要介绍机器学习的基本概念、原理、方法和应用。本书可作为高等院校计算机专业及相关专业的教材，也可供从事机器学习工作的工程技术人员参考。

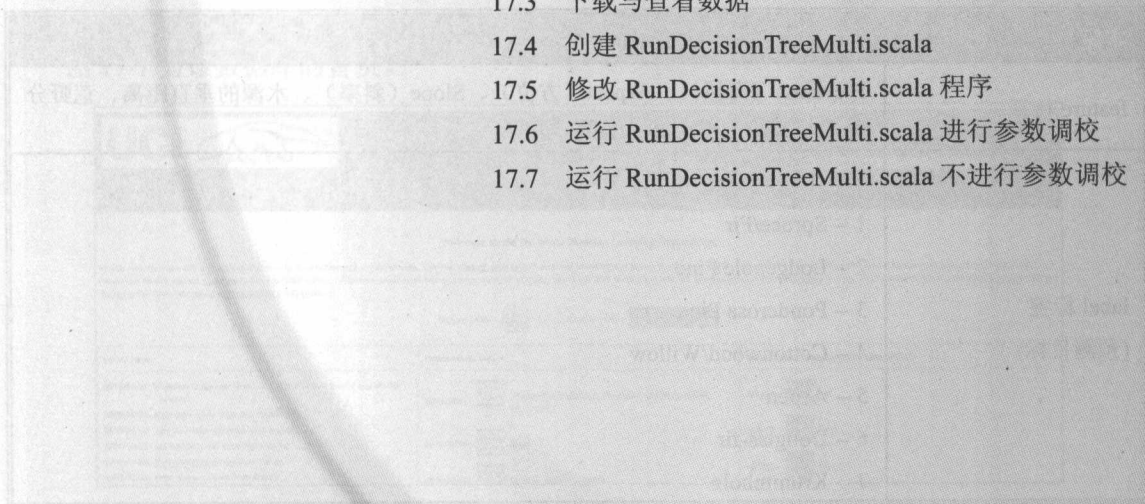
第 17 章 决策树多元分类

第 17 章

决策树多元分类

UCI Covertypes

- 17.1 “森林覆盖植被”大数据问题分析场景
- 17.2 UCI Covertypes 数据集介绍
- 17.3 下载与查看数据
- 17.4 创建 RunDecisionTreeMulti.scala
- 17.5 修改 RunDecisionTreeMulti.scala 程序
- 17.6 运行 RunDecisionTreeMulti.scala 进行参数调校
- 17.7 运行 RunDecisionTreeMulti.scala 不进行参数调校



UCI Covertypes 数据集

该数据集包含 366 个样本，每个样本代表一个森林覆盖的图像。每个样本包含 10 个属性：'CoverType'（覆盖类型）、'SoilType'（土壤类型）、'Aspect'（方位）、'Slope'（坡度）、'Orientation'（朝向）、'Elevation'（海拔）、'HorizontalDistance'（水平距离）、'VerticalDistance'（垂直距离）、'HorizontalDistanceToHilltop'（到山顶的水平距离）和 'VerticalDistanceToHilltop'（到山顶的垂直距离）。'CoverType' 属性是目标变量，其取值范围是 1 到 19。

本章将介绍决策树多元分类（DecisionTree Multi-Class Classification），用于预测“森林覆盖植被”。

17.1 “森林覆盖植被”大数据问题分析场景

森林的管理单位希望能运用大数据分析，帮助他们可以更节省人力、经费来管理森林，更加提高森林覆盖率。因此找来了大数据分析师，与森林管理专员组成一个团队，负责“森林覆盖植被”大数据项目。

➤ 找出问题

“问对问题”是解决问题的第一步。大数据分析师与森林管理专员讨论后发现，在一处森林中有各种树种，例如：热带、针叶林……，每一块土地有它适合生长的植被。如果能够预测哪些土地适合生长的哪些植被，就能在适合的土地种植适当的植被，这样就可以节省人力、经费而达到事半功倍的效果，森林也可以长得更好。

➤ 设计解决方案模型

大数据分析师与森林管理专员讨论哪些因素可能会影响适合生长的植被，经过讨论结果认为：

Elevation（海拔）、Aspect（方位）、Slope（斜率）、水源的垂直距离、荒野分类、水源的水平距离、土壤分类等，都会影响适合生长的植被。因此大数据分析师整理数据如下表格：

字段	说明
feature 特征	Elevation（海拔）、Aspect（方位）、Slope（斜率）、水源的垂直距离、荒野分类、水源的水平距离、土壤分类等
label 标签 (预测目标)	Cover Type 森林覆盖分类 1 - Spruce/Fir 2 - Lodgepole Pine 3 - Ponderosa Pine 4 - Cottonwood/Willow 5 - Aspen 6 - Douglas-fir 7 - Krummholz

➤ 搜集数据

这些数据的搜集可能需要花费很多时间和人力，不过通常森林管理单位已经有长期田野调查的数据，并且借助现代科技，例如无人机空拍、卫星图等，可以搜集这些数据。

➤ 分析数据

大数据分析师决定使用决策树多元分类 (DecisionTree Multi-Class Classification) 来创建模型、训练、评估、预测数据。本章将详细介绍如何进行决策树多元分类分析。

➤ 其他类似的分类应用

大家可能会觉得这个主题“森林覆盖植被”除了森林管理的专业外,其他日常生活可能应用不到。事实上这种类似的应用也很广泛,例如可以研究什么样的地点适合开设什么类型的店面,我们可以搜集相关数据:

- feature 特征: 距离快捷公交站的距离、距离大学的距离、距离中学的距离、该地平均年收入、马路宽度、附近人口数、门口每小时人流量……
- label 标签 (预测目标): 配合实际调查该地店面的类型,已经 3 年以上并且赚钱的店面类型 (代表该地适合这种类型的店面)。

有了这些数据就可以创建模型、训练、评估,预测该地适合开设的店面类型。

17.2 UCI Covertypes 数据集介绍

步骤 01 Machine Learning Repository 数据库

Machine Learning Repository 数据库是加州大学尔湾分校 (University of California Irvine) 提供的用于研究机器学习的数据库。数据库数量与种类繁多,还在不断增加。它的网址为: <http://archive.ics.uci.edu/ml/>。

图 17-1 为该数据库的首页。

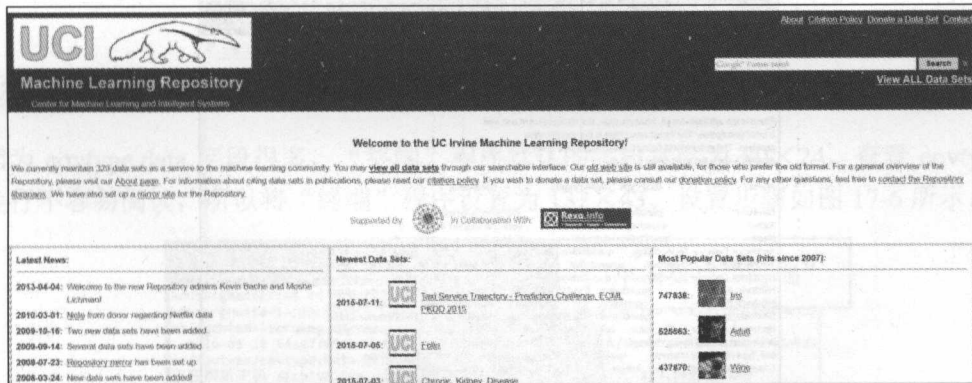


图 17-1 Machine Learning Repository 数据库

步骤 02 UCI Covertypes 数据集

其中具有 Covertypes 数据集, 如下列网址: <https://archive.ics.uci.edu/ml/datasets/Covertypes>
Covertypes Data Set 森林覆盖植被数据集, 是在一处森林中有各种树种植被, 例如, 热带、针叶林等。图 17-2 为该数据集在网页上的下载链接。

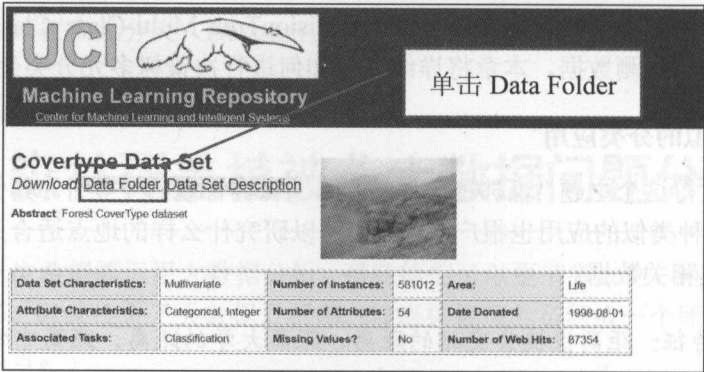


图 17-2 UCI Covertype 数据集的下载链接

步骤 03 查看 Data Folder (见图 17-3)

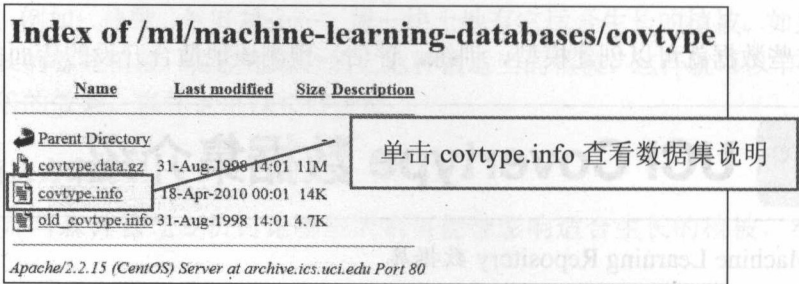


图 17-3 查看 Data Folder

步骤 04 查看 covtype.info

covtype.info 数据集说明如图 17-4 所示。

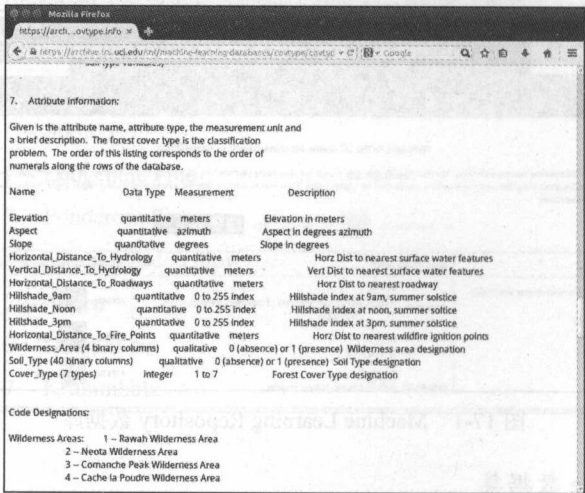


图 17-4 查看 covtype.info

17.3 下载与查看数据

步骤 01 下载/解压缩文件

在“终端”程序中输入下列命令：

➤ 切换到项目数据目录

```
cd ~/workspace/Classification/data
```

➤ 下载 covtype.data.gz

```
wget https://archive.ics.uci.edu/ml/machine-learning-databases/covtype/covtype.data.gz
```

➤ 解压缩到当前目录

```
gzip -d covtype.data.gz
```

运行后屏幕显示界面如图 17-5 所示。

```
hduser@master: ~/workspace/Classification/data
hduser@master:~$ cd ~/workspace/Classification/data
hduser@master:~/workspace/Classification/data$ wget https://archive.ics.uci.edu/
ml/machine-learning-databases/covtype/covtype.data.gz
--2016-05-18 14:47:34-- https://archive.ics.uci.edu/ml/machine-learning-databases/covtype/covtype.data.gz
正在解析主机 archive.ics.uci.edu (archive.ics.uci.edu)... 128.195.10.249
正在连接 archive.ics.uci.edu (archive.ics.uci.edu)[128.195.10.249]:443... 已连接
已发出 HTTP 请求，正在等待回应... 200 OK
长度：11240707 (11M) [application/x-gzip]
正在保存至：“covtype.data.gz”
100%[=====] 11,240,707 22.1KB/s 用时 5m 52s
2016-05-18 14:53:30 (31.2 KB/s) - 已保存“covtype.data.gz” [11240707/11240707]
hduser@master:~/workspace/Classification/data$ gzip -d covtype.data.gz
```

图 17-5 下载/解压缩文件

步骤 02 设置“终端”程序的显示宽度

因为 covtype.data 字段很多，“终端”程序默认的字段宽度为 80×24，查看 covtype.data 时会换行不容易阅读，所以将“终端”程序设置为 132×43。设置步骤如图 17-6 所示。

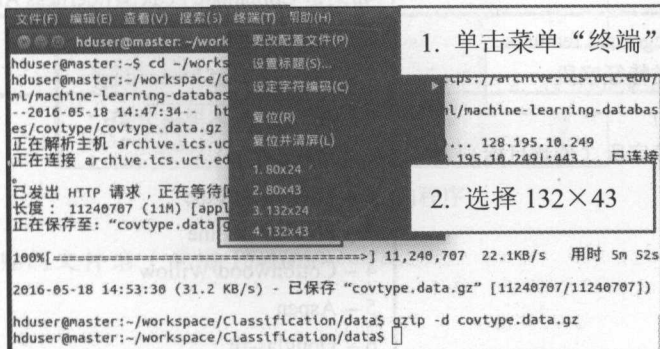


图 17-6 设置“终端”程序的显示宽度

步骤 03 查看 covtype.data 数据

使用下述命令，查看 covtype.data:

```
cat covtype.data | more
```

运行后屏幕显示界面如图 17-7 所示。

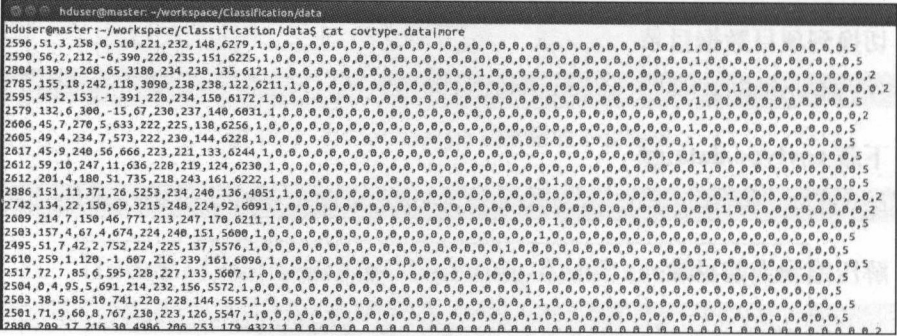


图 17-7 查看 covtype.data 数据

字段说明如下:

字段	字段分类	说明
字段 1~10	Numerical features 数值特征字段	数值字段: 例如 Elevation (海拔)、Aspect (方位)、Slope (斜率)、Vertical_Distance_To_Hydrology (距离水源的垂直距离)、Horizontal_Distance_To_Roadways (距离水源的水平距离)、Hillshade_9am (9 点时阴影) 等
字段 11~14	Categorical features 分类特征字段	Wilderness Areas 荒野分类字段: 1 – Rawah Wilderness Area 2 – Neota Wilderness Area 3 – Comanche Peak Wilderness Area 4 – Cache la Poudre Wilderness Area 已经编码成 1-of-k encoding。 例如: 如果是 Neota Wilderness Area, 则为 0,1,0,0。 如果是 Comanche Peak Wilderness Area, 则为 0,0,1,0
字段 15~54	Categorical features 分类特征字段	Soil Types 土壤分类。 已经编码成 1-of-k encoding
字段 55	label 标签字段 (预测目标)	Cover Type 森林覆盖分类: 1 – Spruce/Fir 2 – Lodgepole Pine 3 – Ponderosa Pine 4 – Cottonwood/Willow 5 – Aspen 6 – Douglas-fir 7 – Krummholz

可以看到上述 Categorical features 分类特征字段：荒野分类、土壤分类，都已经是编码成 1-of-k encoding 的数字字段，所以不需要自己转换为数字字段。

17.4 创建 RunDecisionTreeMulti.scala

因为决策树多元分析 RunDecisionTreeMulti.scala 的程序基本架构，与第 13 章介绍的决策树二元分析 RunDecisionTreeBinary.scala 类似。所以将复制 RunDecisionTreeBinary.scala 到 RunDecisionTreeMulti.scala 中进行修改。

步骤 01 复制 RunDecisionTreeBinary.scala（见图 17-8）

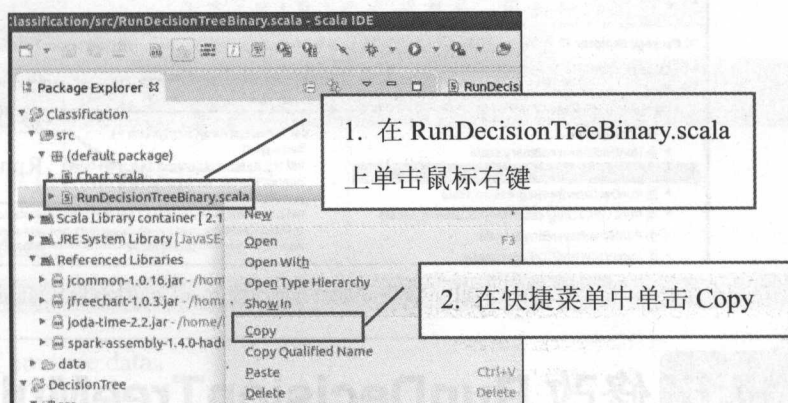


图 17-8 复制 RunDecisionTreeBinary.scala

步骤 02 粘贴复制的程序（见图 17-9）

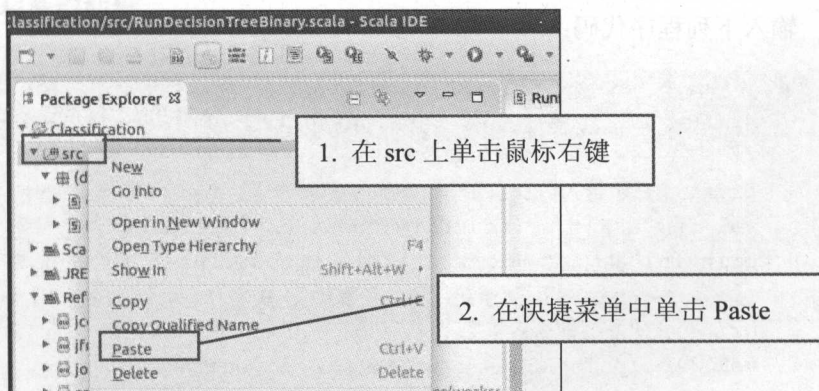


图 17-9 粘贴复制的程序

步骤 03 输入要粘贴的文件名（见图 17-10）

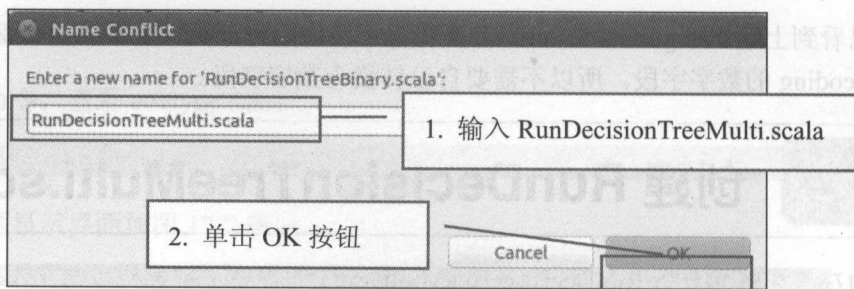


图 17-10 输入要粘贴的文件名

步骤 04 修改 object name (见图 17-11)

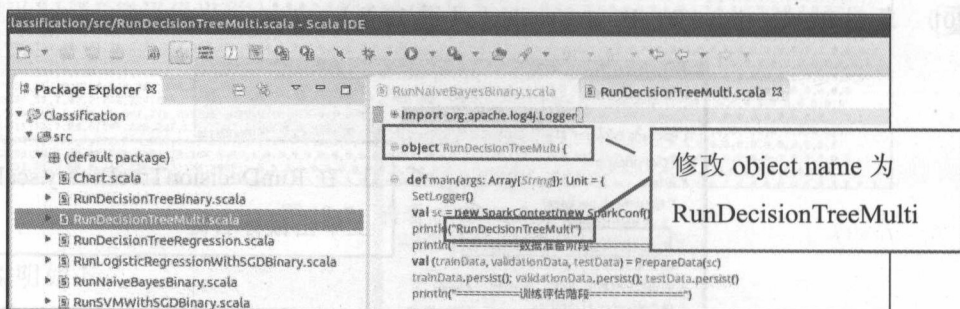


图 17-11 修改 object name 为 RunDecisionTreeMulti

17.5 修改 RunDecisionTreeMulti.scala 程序

步骤 01 修改 PrepareData()数据准备

输入下列程序代码:

```
def PrepareData(sc: SparkContext):
(RDD[LabeledPoint], RDD[LabeledPoint], RDD[LabeledPoint]) = {
  //-----1.导入转换数据-----
  print("开始导入数据...")
  val rawData = sc.textFile("data/covtype.data")
  println("共计: " + rawData.count.toString() + "条")
  //-----2.创建训练评估所需数据 RDD[LabeledPoint]-----
  println("准备训练数据...")
  val labelpointRDD = rawData.map { record =>
    val fields = record.split(',').map(_.toDouble)
    val label = fields.last - 1
    LabeledPoint(label, Vectors.dense(fields.init))
  }
  println(labelpointRDD.first())
  //-----3.以随机方式将数据分为3个部分并且返回-----
  val Array(trainData, validationData, testData) =
```



```
labelpointRDD.randomSplit(Array(0.8, 0.1, 0.1))
println("将数据分为 trainData:" + trainData.count() +
" cvData:" + validationData.count() +
" testData:" + testData.count())
return (trainData, validationData, testData)
}
```

➤ 定义 PrepareData 函数

```
def PrepareData(sc: SparkContext): (RDD[LabeledPoint],
RDD[LabeledPoint], RDD[LabeledPoint]) = {
```

PrepareData 函数传入参数 sc 是 SparkContext; 返回: trainData、validationData、testData。

➤ 创建 SparkContext

```
val sc = new SparkContext(new SparkConf()
.setAppName("DecisionTreeMulti").setMaster("local[4]"))
```

创建 SparkContext, 默认在本地运行。

➤ 读取 covtype.data

```
val rawData = sc.textFile("data/covtype.data")
```

使用 sc.textFile 读取 covtype.data。

➤ 输出数据条数

```
println("共计: " + rawData.count.toString() + "条")
```

➤ 使用 map 进行数据转换

```
val labelpointRDD= rawData.map { record =>
...
}
```

使用 map 传入 record 作为参数, 针对每一条数据进行转换。

➤ 提取每一个字段

```
val fields = record.split(',').map(_.toDouble)
```

record.split(',') 以 “,” 分隔字段, 提取每个字段并转换为 Double。

➤ 提取 label 字段

```
val label = fields.last- 1
```

label 字段使用 fields.last 读取最后一个字段。

- 原本 label 的值范围是 1~7 (1— Spruce/Fir、2— Lodgepole Pine、3— Ponderosa Pine、

4—Cottonwood/Willow、5—Aspen、6—Douglas-fir、7—Krummholz)。

- 但是, 因为要进行 DecisionTree.trainClassifier 训练, 规定 label 值一定要从 0 开始的数值, 所以必须使用 “fields.last-1” 减 1。label 值范围变成是 0~6。

➤ 创建 LabelPoint

```
LabeledPoint(label, Vectors.dense(fields.init))
```

创建 LabelPoint 是由标签字段与特征字段组成:

- 标签字段: label。
- 特征字段: 使用 fields.init 读取除了最后字段的所有字段。

➤ 以随机方式将数据分为 3 个部分

```
val Array(trainData, validationData, testData) =  
  labelpointRDD.randomSplit(Array(0.8, 0.1, 0.1))
```

➤ 显示数据记录数

```
println("将数据分为 trainData:" + trainData.count() +  
  " cvData:" + validationData.count() + " testData:" + testData.count())
```

➤ 返回数据

```
return (trainData, validationData, testData)
```

步骤 02 修改 trainModel 训练模型 (见图 17-12)

```
def trainModel(trainData: RDD[LabeledPoint], impurity: String, maxDepth: Int, maxBins: Int): (DecisionTreeModel, Double) = {  
  val startTime = new DateTime()  
  val model = DecisionTree.trainClassifier(trainData, 7, Map[Int, Int](), impurity, maxDepth, maxBins)  
  val endTime = new DateTime()  
  val duration = new Duration(startTime, endTime)  
  (model, duration.getMillis())  
}
```

numClasses 原本为 2 修改为 7

图 17-12 修改 trainModel 训练模型

➤ 决策树二元分类 numClasses 设置

在之前的决策树二元分类 RunDecisionTreeBinary.scala 中, 因为 Label 标签字段只有 0 或 1, 所以分类数目 numClasses 设置为 2。

```
val model = DecisionTree.trainClassifier(trainData, 2,  
  Map[Int, Int](), impurity, maxDepth, maxBins)
```

➤ 决策树多元分类 numClasses 设置

但是, 在决策树多元分类中, 我们的 Label 标签字段有 7 种可能值, 所以分类数目 numClasses 设置为 7。

```
val model = DecisionTree.trainClassifier(trainData, 7,
Map[Int, Int](), impurity, maxDepth, maxBins)
```

步骤 03 修改 evaluateModel 评估模型（见图 17-13）

```
*RunDecisionTreeMulti.scala
def evaluateModel(model: DecisionTreeModel, validationData: RDD[LabeledPoint]): (Double) = {
  val scoreAndLabels = validationData.map { data =>
    var predict = model.predict(data.features)
    (predict, data.label)
  }
  val Metrics = new MulticlassMetrics(scoreAndLabels)
  val precision = Metrics.precision
  (precision)
}
```

改为 MulticlassMetrics

图 17-13 修改 evaluateModel 评估模型

二元分类评估方式

在之前的决策树二元分类 RunDecisionTreeBinary.scala 中，我们是以 AUC 作为评估模型的准确率。所以使用 BinaryClassificationMetrics 先创建 Metrics，然后使用.areaUnderROC 方法计算 AUC，如下程序：

```
val Metrics = new BinaryClassificationMetrics(scoreAndLabels)
val AUC = Metrics.areaUnderROC
(AUC)
```

多元分类评估方式

但是在多元分类 (Multiclass Classification) 中，我们使用 MulticlassMetrics 先创建 Metrics，然后使用.precision 方法计算准确率，如下程序：

```
val Metrics = new MulticlassMetrics(scoreAndLabels)
val precision = Metrics.precision
(precision)
```

步骤 04 修改 evaluateParameter 绘图程序代码（见图 17-14）

```
def evaluateParameter(trainData: RDD[LabeledPoint], validationData: RDD[LabeledPoint],
  evaluateParameter: String, impurityArray: Array[String], maxdepthArray: Array[Int], maxBinsArray: Array[Int]) =
{
  var dataBarChart = new DefaultCategoryDataset()
  var dataLineChart = new DefaultCategoryDataset()
  for (impurity <- impurityArray; maxDepth <- maxdepthArray; maxBins <- maxBinsArray) {
    val (model, time) = trainModel(trainData, impurity, maxDepth, maxBins)
    val precise = evaluateModel(model, validationData)
    val parameterData =
      evaluateParameter match {
        case "impurity" => impurity;
        case "maxDepth" => maxDepth;
        case "maxBins" => maxBins
      }
    dataBarChart.addValue(precise, evaluateParameter, parameterData.toString())
    dataLineChart.addValue(time, "Time", parameterData.toString())
  }
  Chart.plotBarLineChart("DecisionTree evaluations " + evaluateParameter, evaluateParameter, "precision", 0.6, 1, "Time", dataBarChart, dataLineChart)
}
```

修改"precision", 0.6, 1

图 17-14 修改 evaluateParameter 绘图程序代码

在多元分类中，我们使用 `precision` 评估模型的准确率。因为数值的范围不同，修改显示数值柱形图的显示范围如下：

```
Chart.plotBarLineChart("DecisionTree evaluations " + evaluateParameter,
    evaluateParameter, "precision", 0.6, 1, "Time", dataBarChart, dataLineChart)
```

步骤 05 输入 PredictData 程序代码

输入 PredictData 程序代码如下：

```
def PredictData(sc: SparkContext, model: DecisionTreeModel): Unit = {
    //-----1.导入转换数据-----
    val rawData = sc.textFile("data/covtype.data")
    println("共计: " + rawData.count.toString() + "条")
    //-----2.创建训练评估所需数据 RDD[LabeledPoint]-----
    println("准备训练数据...")
    val Array(pData, oData) = rawData.randomSplit(Array(0.1, 0.9))
    val data = pData.take(20).map { record =>
        val fields = record.split(',').map(_.toDouble)
        val features = Vectors.dense(fields.init)
        val label = fields.last - 1
        val predict = model.predict(features)
        val result = (if (label == predict) "正确" else "错误")
        println("土地条件: 海拔:" + features(0) + " 方位:" + features(1) +
            " 斜率:" + features(2) + " 水源垂直距离:" + features(3) +
            " 水源水平距离:" + features(4) + " 9点时阴影:" + features(5) +
            "....=>预测:" + predict + " 实际:" + label + "结果:" + result)
    }
}
```

PredictData 与 PrepareData 程序代码类似，以下仅说明不同之处。

➤ 定义 PredictData 函数

```
def PredictData(sc: SparkContext, model: DecisionTreeModel): Unit = {
```

PredictData 函数传入参数 `sc` 是 `SparkContext`；返回: `model: DecisionTreeModel` 训练完成的模型。

➤ 读取预测的数据

```
val rawData = sc.textFile("data/covtype.data")
```

因为此数据集并没有提供预测的数据，所以仍然使用 `covtype.data` 进行预测。

➤ 随机数读取测试数据

```
val Array(pData, oData) = rawData.randomSplit(Array(0.1, 0.9))
```


使用 randomSplit 随机数读取测试数据。

➤ 读取 20 条进行预测

```
val data = pData.take(20).map { line =>
```

使用 pData 使用 take(20) 读取 20 条数据，使用 map 进行转换。

➤ 进行预测

```
val predict = model.predict(feature)
```

使用 model.predict 传入 feature 参数进行预测。

➤ 对比实际与预测数据

```
val result = (if (label == predict) "正确" else "错误")
```

label 标签字段是实际的数据，predict 是预测的结果，对比之后得出“正确”或“错误”。

➤ 卷标输出预测结果

```
println("土地条件: 海拔:" + features(0) + " 方位:" + features(1) +  
" 斜率:" + features(2) + " 水源垂直距离:" + features(3) +  
" 水源水平距离:" + features(4) + " 9点时阴影:" + features(5) +  
"....==>预测:" + predict + " 实际:" + label + "结果:" + result)
```

17.6

运行 RunDecisionTreeMulti.scala 进行参数调校

步骤 01 依次选择菜单及其菜单项 Run→Run Configurations (见图 17-15)

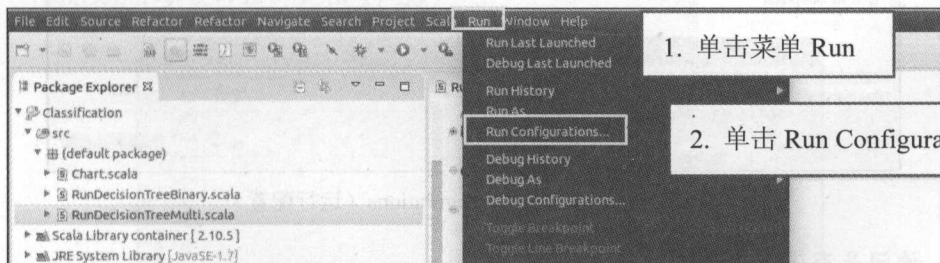


图 17-15 依次选择菜单及其菜单项 Run→Run Configurations

步骤 02 复制 Run Configurations (见图 17-16)

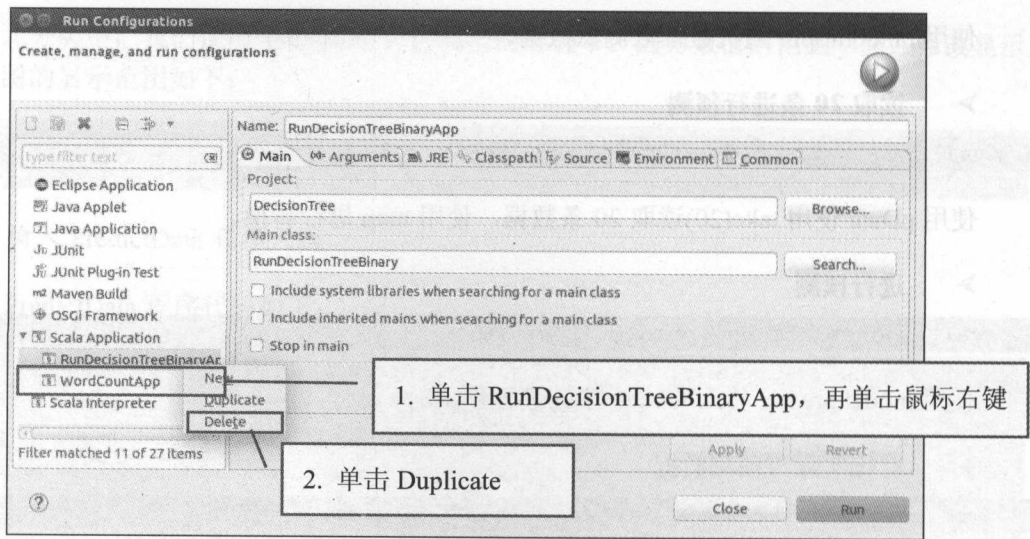


图 17-16 复制 Run Configurations（运行配置）

步骤 03 设置 Run Configurations（见图 17-17）

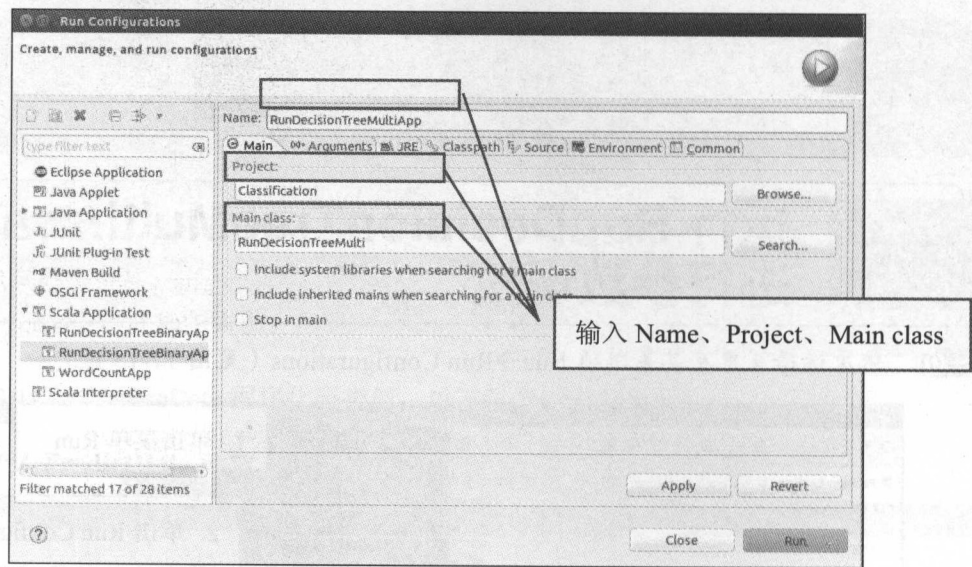


图 17-17 设置 Run Configurations（运行配置）

步骤 04 询问是否进行参数调校

程序会询问是否需要进行参数调校（Y：是、N：否）？输入 Y。如图 17-18 所示。



图 17-18 选择进行参数调校

步骤 05 Impurity 参数评估 (见图 17-19)

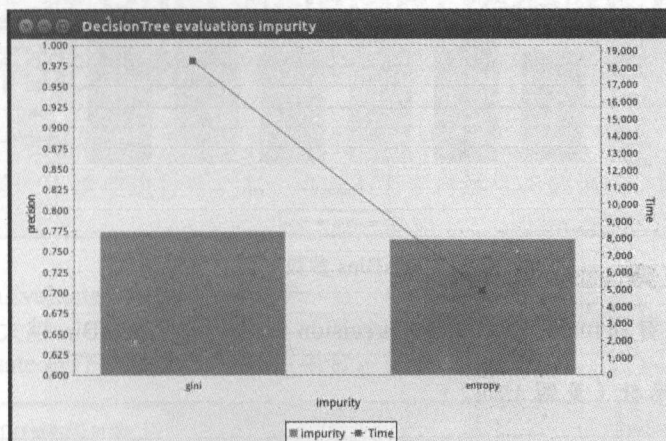


图 17-19 Impurity 参数评估的结果图

柱形图代表 precise、折线图代表时间。从图 17-19 可以看到 gini (基尼) 准确度比 entropy (熵) 好一些, 但是并没有很大的差别。但是运行时间 gini 所需的时间是 entropy 的 4 倍。

步骤 06 maxDepth 参数评估 (见图 17-20)

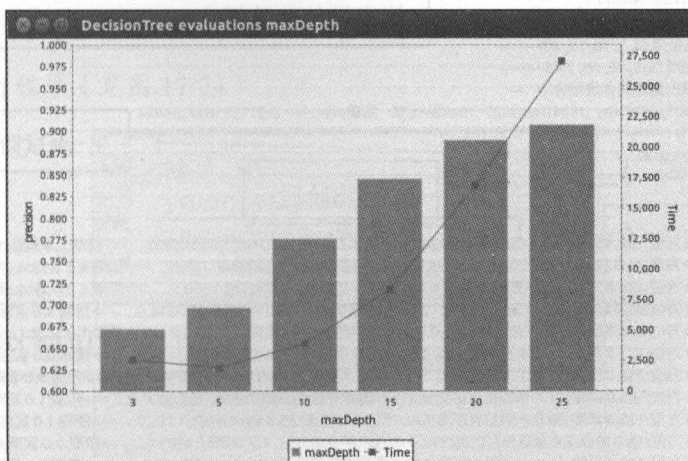


图 17-20 maxDepth 参数评估的结果图

从图 17-20 可以看到 maxDepth= 25 时, precision 最高。随着 maxDepth 越大, 所需的时间越多。

步骤 07 maxBins 参数评估 (见图 17-21)

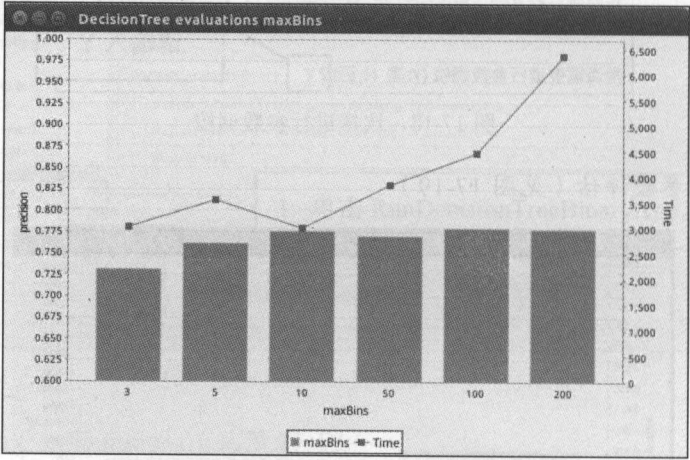


图 17-21 maxBins 参数评估的结果图

从图 17-21 可以看到 maxBin=100 时, precision 最高。随着 maxBin 越大, 所需的时间越多。

步骤 08 评估所有参数 (见图 17-22)

ProblemsTasksConsole

RunDecisionTreeMultiApp [Scala Application] /usr/lib/ivm/java-7-openjdk-amd64/bin/java (2016年5月18日 下午9:30:27)
RunDecisionTreeMulti
=====数据准备阶段=====
开始导入数据...共计: 581012条
准备训练数据...
将数据分为 trainData:464154 cvData:58514 testData:58514
=====训练评估阶段=====
是否需要参数调校 (Y:是 N:否) ? Y
——评估 Impurity 参数使用 gini, entropy——
——评估 MaxDepth 参数使用 (3, 5, 10, 15, 20)——
——评估 maxBins 参数使用 (3, 5, 10, 50, 100)——
——所有参数交叉评估找出最好的参数组合——
调校后最佳参数: impurity:entropy ,maxDepth:20 ,maxBins:50 ,结果precision = 0.911371637556824
=====测试阶段=====
使用testata测试,结果 precision:0.9170608802961744
=====预测数据=====
共计: 581012条
准备测试数据...
土地条件: 海拔:2579.0 方位:132.0 斜率:6.0 水源垂直距离:300.0 水源水平距离:-15.0 9点时阴影:67.0...==>预测:1.0 实际:1.0 结果:正确
土地条件: 海拔:2501.0 方位:71.0 斜率:9.0 水源垂直距离:60.0 水源水平距离:8.0 9点时阴影:767.0...==>预测:4.0 实际:4.0 结果:正确
土地条件: 海拔:2492.0 方位:135.0 斜率:6.0 水源垂直距离:0.0 水源水平距离:0.0 9点时阴影:860.0...==>预测:4.0 实际:4.0 结果:正确
土地条件: 海拔:2749.0 方位:98.0 斜率:30.0 水源垂直距离:124.0 水源水平距离:53.0 9点时阴影:3316.0...==>预测:4.0 实际:4.0 结果:正确
土地条件: 海拔:2570.0 方位:346.0 斜率:2.0 水源垂直距离:0.0 水源水平距离:0.0 9点时阴影:331.0...==>预测:1.0 实际:1.0 结果:正确
土地条件: 海拔:2739.0 方位:323.0 斜率:25.0 水源垂直距离:85.0 水源水平距离:43.0 9点时阴影:3118.0...==>预测:0.0 实际:0.0 结果:正确
土地条件: 海拔:2860.0 方位:358.0 斜率:17.0 水源垂直距离:175.0 水源水平距离:98.0 9点时阴影:3705.0...==>预测:0.0 实际:0.0 结果:正确
土地条件: 海拔:3182.0 方位:105.0 斜率:11.0 水源垂直距离:408.0 水源水平距离:33.0 9点时阴影:6000.0...==>预测:1.0 实际:0.0 结果:错误
土地条件: 海拔:2644.0 方位:125.0 斜率:20.0 水源垂直距离:67.0 水源水平距离:25.0 9点时阴影:1719.0...==>预测:1.0 实际:1.0 结果:正确
土地条件: 海拔:2886.0 方位:12.0 斜率:7.0 水源垂直距离:570.0 水源水平距离:94.0 9点时阴影:4295.0...==>预测:0.0 实际:0.0 结果:正确
土地条件: 海拔:3034.0 方位:189.0 斜率:14.0 水源垂直距离:384.0 水源水平距离:66.0 9点时阴影:5754.0...==>预测:0.0 实际:0.0 结果:正确

1. 数据准备阶段

2. 训练评估阶段, 进行参数调校
找出最佳参数组合

3. 测试阶段

4. 预测阶段

图 17-22 评估所有参数

1. **数据准备阶段**: 显示导入个数以及 trainData validationData testData 个数。
2. **训练评估阶段**: 进行参数调校。调校后最佳参数: impurity:entropy、maxDepth:20、maxBins:50, 结果 precise = 0.91。
找出最佳参数组合: precise 大约 0.91。
3. **测试阶段**: 使用测试数据再测试模型, precise 大约 0.91。与训练阶段差异不大, 确认没有 overfitting 的问题。
4. **预测阶段**: 使用 covtype.data 预测, 显示在不同的土地条件下, 预测的值与实际的值, 并显示预测结果是否正确。

17.7 运行 RunDecisionTreeMulti.scala 不进行参数调校

因为参数调校比较花时间, 所以当我们已经找出最佳参数组合, 就可以修改 trainEvaluate 程序为最佳参数组合。下次要进行预测时, 可以不需要再运行参数调校, 直接使用最佳参数进行预测即可。

步骤 01 修改 trainEvaluate 为最佳参数组合

修改 trainEvaluate 程序, 改为最佳参数组合。如图 17-23 所示。

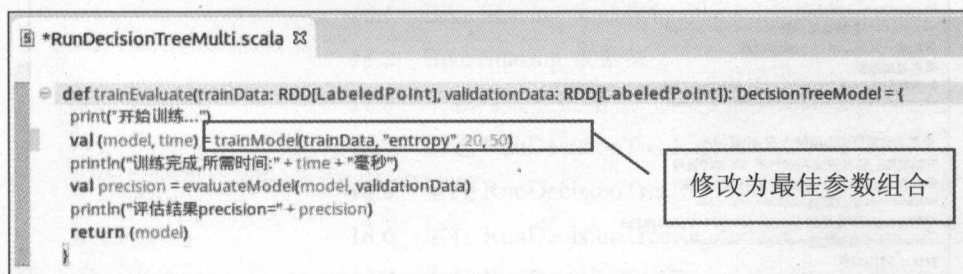


图 17-23 修改 trainEvaluate 为最佳参数组合

步骤 02 开始运行程序 (见图 17-24)

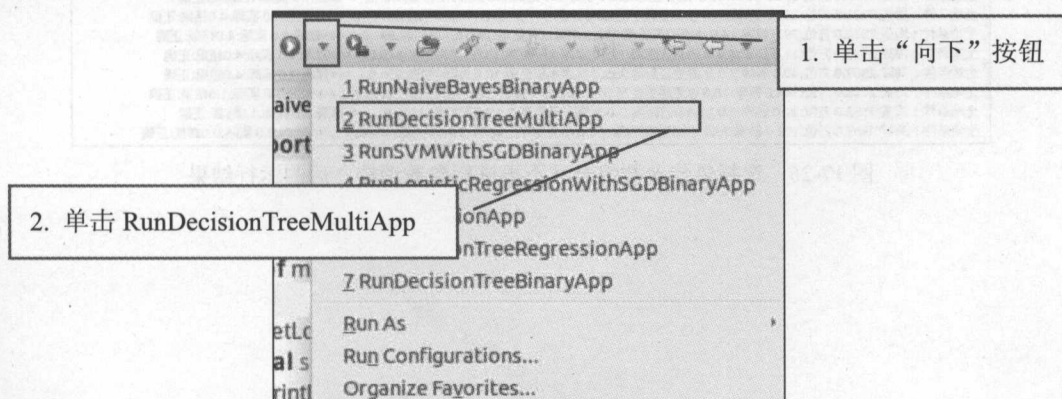


图 17-24 开始运行 RunDecisionTreeMultiApp 程序

步骤 03 不需要进行参数调校

再次运行程序，程序会询问是否需要进行参数调校（Y：是、N：否）？输入 N。如图 17-25 所示。

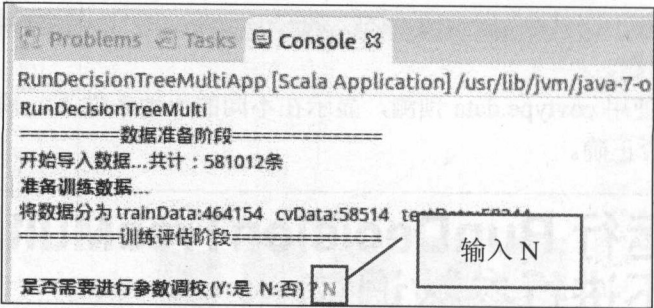


图 17-25 选择不进行参数调校

步骤 04 再次运行程序

运行结果如图 17-26 所示。可以看到程序训练完成后就可以进行预测。因为不进行参数调校，所以运行所需的时间比较少。

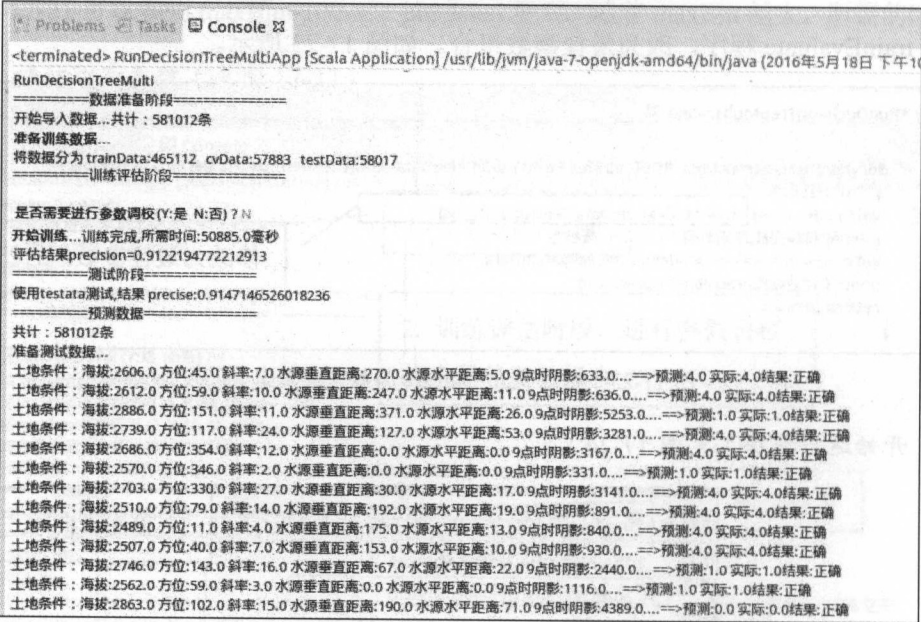


图 17-26 选择最佳参数组合且不再进行参数调校之后的运行结果

第 18 章

决策树回归分析

- 18.1 Bike Sharing 大数据问题分析
- 18.2 Bike Sharing 数据集
- 18.3 下载与查看数据
- 18.4 创建 RunDecisionTreeRegression.scala
- 18.5 修改 RunDecisionTreeRegression.scala
- 18.6 运行 RunDecisionTreeRegression. scala 进行参数调校
- 18.7 运行 RunDecisionTreeRegression. scala 不进行参数调校

本章将介绍决策树回归分析 (DecisionTree Regression)，用于预测 Bike Sharing 的租用数量。

18.1 Bike Sharing 大数据问题分析

Bike Sharing 系统就像城市的自行车出租系统一样，可以在某一站租借，在另一站归还。租车运营公司的管理层希望能运用大数据分析提供更好的服务。因此，找来了大数据分析师，与公司实际负责运营的工作人员组成一个团队，负责 Bike Sharing 大数据项目。

➤ 找出问题

“问对问题”是解决问题的第一步。大数据分析师与运营人员讨论后发现，运营人员面临一个问题，在不同的天气状态下，租用的数量差异很大。这就造成了困扰，运营人员常常无法提供足够数量的自行车，或是不知何时进行维修工作。

因此，他们希望能够预测在某种天气状态下的自行车租用数量。预知自行车租用数量的好处如下：

- 自行车需要维修时，可以选择在自行车租用数量较少的时段。
- 在自行车租用数量大的时间，可以提供更多的自行车，增加营业额。

➤ 设计解决方案模型

大数据分析师与 Bike Sharing 有实际工作经验的人员讨论：哪些因素可能影响自行车租用的数量，经过讨论结果认为：

季节、月份、时间 (0~23)、假日、星期、工作日、天气、温度、体感温度、湿度、风速，都会影响自行车租借的数量。因此大数据分析师整理数据如下表格：

字段	说明
feature 特征	季节、月份、时间 (0~23)、假日、星期、工作日、天气、温度、体感温度、湿度、风速
label 标签 (预测目标)	每一个小时租用数量

➤ 搜集数据

- 每小时自行车租用数量可以在 Bike Sharing 公司的计算机系统查询。
- 季节、月份、时间 (0~23)、假日、星期、工作日，程序系统可以由每小时租用数量自动算出。
- 天气、温度、体感温度、湿度、风速这些天气的数据，可以与提供天气历史数据的公司合作来获得。

➤ 分析数据

大数据分析师决定使用决策树回归分析来创建模、训练、评估、预测数据。本章将详细介绍如何进行决策树回归分析。

➤ 其他根据天气的预测应用

事实上,这种根据天气预测的应用很广泛。例如,小连锁超市可以根据天气来预测关东煮、茶叶蛋、雨衣等的销售数量。如果可以预测销售数量,就可以事先准备足够的商品数量,从而增加销售额。也可以减少因为准备过多的商品,而导致食材的浪费或者商品的积压。除了小连锁超市,餐厅也很适合使用天气来预测销售数量。

18.2 Bike Sharing 数据集

不过在本章中不需要自行搜集这些数据,已经有研究单位已经创建了这些数据。可以使用浏览器输入下列网址来下载这些数据: <https://archive.ics.uci.edu/ml/datasets/Bike+Sharing+Dataset>
图 18-1 即为 Bike Sharing 数据集的下载网站。

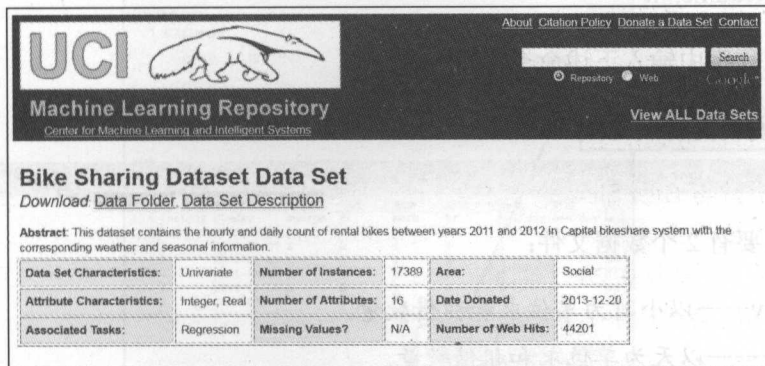


图 18-1 Bike Sharing 数据集的下载网站

18.3 下载与查看数据

步骤 01 下载/解压缩文件

在“终端”程序中输入下述命令:

➤ 切换到项目数据目录

```
cd ~/workspace/Classification/data
```

➤ 下载 Bike-Sharing-Dataset.zip

```
wget https://archive.ics.uci.edu/ml/machine-learning-databases/00275/Bike-Sharing-Dataset.zip
```

➤ 解压缩 Bike-Sharing-Dataset.zip

```
unzip -j Bike-Sharing-Dataset.zip
```

运行后屏幕显示界面如图 18-2 所示。

```
hduser@master: ~/workspace/Classification/data
hduser@master:~$ cd ~/workspace/Classification/data
hduser@master:~/workspace/Classification/data$ wget https://archive.ics.uci.edu/
ml/machine-learning-databases/00275/Bike-Sharing-Dataset.zip
--2016-05-18 22:14:17-- https://archive.ics.uci.edu/ml/machine-learning-databases/00275/Bike-Sharing-Dataset.zip
正在解析主机 archive.ics.uci.edu (archive.ics.uci.edu)... 128.195.10.249
正在连接 archive.ics.uci.edu (archive.ics.uci.edu)[128.195.10.249]:443... 已连接
已发出 HTTP 请求, 正在等待回应... 200 OK
长度: 279992 (273K) [application/zip]
正在保存至: "Bike-Sharing-Dataset.zip"

100%[=====] 279,992 44.0KB/s 用时 6.2s

2016-05-18 22:14:25 (44.0 KB/s) - 已保存 "Bike-Sharing-Dataset.zip" [279992/2799
92]]

hduser@master:~/workspace/Classification/data$ unzip -j Bike-Sharing-Dataset.zip
Archive: Bike-Sharing-Dataset.zip
  inflating: Readme.txt
  inflating: day.csv
  inflating: hour.csv
hduser@master:~/workspace/Classification/data$
```

图 18-2 下载/解压缩文件

步骤 02 查看 Readme.txt

在“终端”程序中输入下述命令:

➤ 查看说明文件 (见图 18-3)

```
cat Readme.txt|more
```

在目录下主要有 2 个数据文件:

- hour.csv——以小时为单位求和租借数量。
- day.csv——以天为单位求和租借数量。

```
hduser@master: ~/workspace/DecisionTree/data
files
- Readme.txt
- hour.csv - bike sharing counts aggregated on hourly basis. Records: 17379 hours
- day.csv - bike sharing counts aggregated on daily basis. Records: 731 days

Dataset characteristics
Both hour.csv and day.csv have the following fields, except hr which is not available in day.csv

- instant: record index
- dteday : date
- season : season (1:springer, 2:summer, 3:fall, 4:winter)
- yr : year (0: 2011, 1:2012)
- mth : month ( 1 to 12)
- hr : hour (0 to 23)
- holiday : weather day is holiday or not (extracted from http://dchr.dc.gov/page/holiday-schedule)
- weekday : day of the week
- workingday : if day is neither weekend nor holiday is 1, otherwise is 0.
+ weathersit :
  - 1: Clear, Few clouds, Partly cloudy, Partly cloudy
  - 2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
  - 3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds
  - 4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
- temp : Normalized temperature in Celsius. The values are divided to 41 (max)
- atemp: Normalized feeling temperature in Celsius. The values are divided to 50 (max)
- hum: Normalized humidity. The values are divided to 100 (max)
- windspeed: Normalized wind speed. The values are divided to 67 (max)
- casual: count of casual users
- registered: count of registered users
- cnt: count of total rental bikes including both casual and registered
```

图 18-3 查看 Readme.txt

步骤 03 双击 hour.csv 打开该文件（见图 18-4）



图 18-4 双击 hour.csv 以打开该文件

步骤 04 查看 hour.csv（见图 18-5）

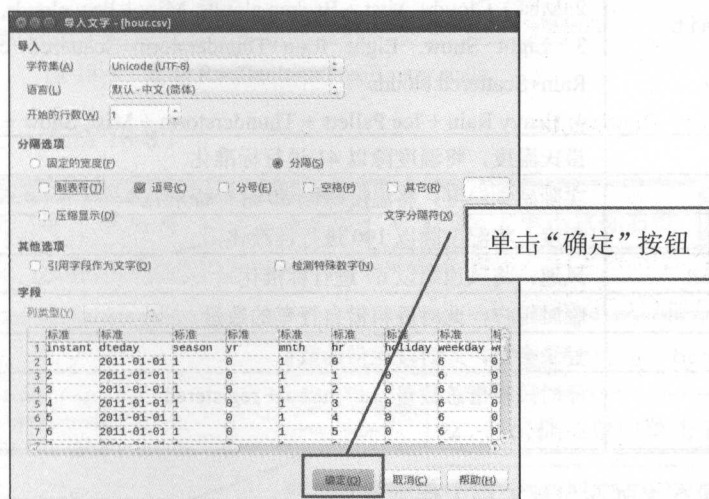


图 18-5 查看 hour.csv

hour.csv 字段介绍（见图 18-6）

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	instant	datetime	season	yr	month	hr	holiday	weekday	workingday	registered	temp	atemp	hum	windspeed	casual	registered	cnt
2	1	2011-01-01	1	0	1	0	0	6	0	1	0.24	0.2879	0.81	0	3	13	16
3	2	2011-01-01	1	0	1	1	0	6	0	1	0.22	0.2727	0.8	0	8	32	40
4	3	2011-01-01	1	0	1	2	0	6	0	1	0.22	0.2727	0.8	0	5	27	32
5	4	2011-01-01	1	0	1	3	0	6	0	1	0.24	0.2879	0.75	0	3	10	13
6	5	2011-01-01	1	0	1	4	0	6	0	1	0.24	0.2879	0.75	0	0	0	1
7	6	2011-01-01	1	0	1	5	0	6	0	2	0.24	0.2576	0.75	0.0896	0	1	1
8	7	2011-01-01	1	0	1	6	0	6	0	1	0.22	0.2727	0.8	0	2	0	2
9	8	2011-01-01	1	0	1	7	0	6	0	1	0.2	0.2576	0.86	0	1	2	3
10	9	2011-01-01	1	0	1	8	0	6	0	1	0.24	0.2879	0.75	0	1	8	8
11	10	2011-01-01	1	0	1	9	0	6	0	1	0.32	0.3485	0.76	0	8	14	14
12	11	2011-01-01	1	0	1	10	0	6	0	1	0.38	0.3939	0.76	0.2537	12	24	36
13	12	2011-01-01	1	0	1	11	0	6	0	1	0.36	0.3333	0.81	0.2836	26	36	56
14	13	2011-01-01	1	0	1	12	0	6	0	1	0.42	0.4242	0.77	0.2836	29	55	84
15	14	2011-01-01	1	0	1	13	0	6	0	2	0.46	0.4545	0.72	0.2985	47	47	94

图 18-6 hour.csv 的各个字段

hour.csv 字段的介绍如下，我们可以先思考一下如何处理。

字段	说明	处理方式
instant	序号 ID	忽略
dteday	日期	忽略
season	季节 (1:springer, 2:summer, 3:fall, 4:winter)	特征字段
yr	年份 (0:2011,1:2012)	忽略
mnth	月份 (1~12)	特征字段
hr	时间 (0~23)	特征字段
holiday	假日: 0:非假日、1:假日	特征字段
weekday	星期	特征字段
workingday	工作日, 非周末而且非假日	特征字段
weathersit	天气: 1: Clear, Few clouds, Partly cloudy, Partly cloudy 2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist 3: Light Snow, Light Rain+Thunderstorm+Scattered clouds,Light Rain+Scattered clouds 4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog	特征字段
temp	摄氏温度, 将温度除以 41 进行标准化	特征字段
atemp	实际感觉温度, 将温度除以 50 进行标准化	特征字段
hum	湿度, 将湿度除以 100 进行标准化	特征字段
windspeed	风速, 将风速除以 67 进行标准化	特征字段
casual	临时用户, 此时段租借自行车的数量	忽略
registered	登录会员, 此时段租借的数目	忽略
cnt	此时段租借总数量 cnt= casual+ registered	标签字段 (预测目标)

以上部分字段会忽略不列入特征字段:

- Instant (序号 ID)、dteday (日期) 与预测的结果无关, 所以忽略这两个字段。
- yr 年份 (0:2011,1:2012), 因为我们希望预测未来的年份, 所以忽略此字段。
- casual 与 registered, 因为 casual 加 registered 等于 cnt (标签字段)。这两个字段已经隐含了标签字段的信息, 所以忽略此字段。

18.4 创建 RunDecisionTreeRegression.scala

因为决策树回归分析 RunDecisionTreeRegression.scala 的程序基本架构, 与第 13 章介绍的决策树二元分析 RunDecisionTreeBinary.scala 类似。所以我们将复制 RunDecisionTreeBinary.scala 到 RunDecisionTreeRegression.scala, 再进行修改。

步骤 01 复制 RunDecisionTreeBinary.scala (见图 18-7)

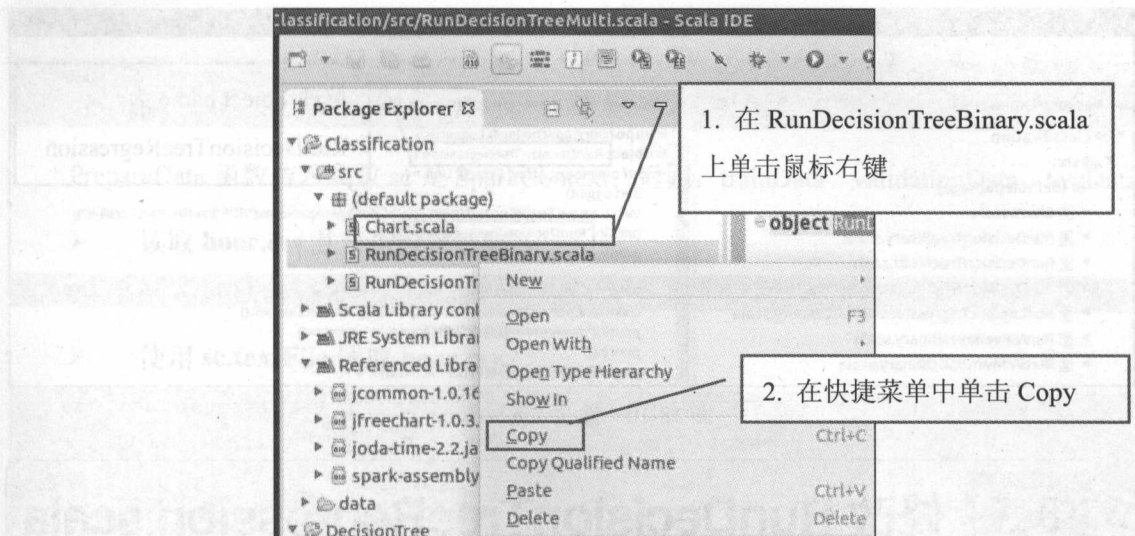


图 18-7 复制 RunDecisionTreeBinary.scala

步骤 02 粘贴复制的程序 (见图 18-8)

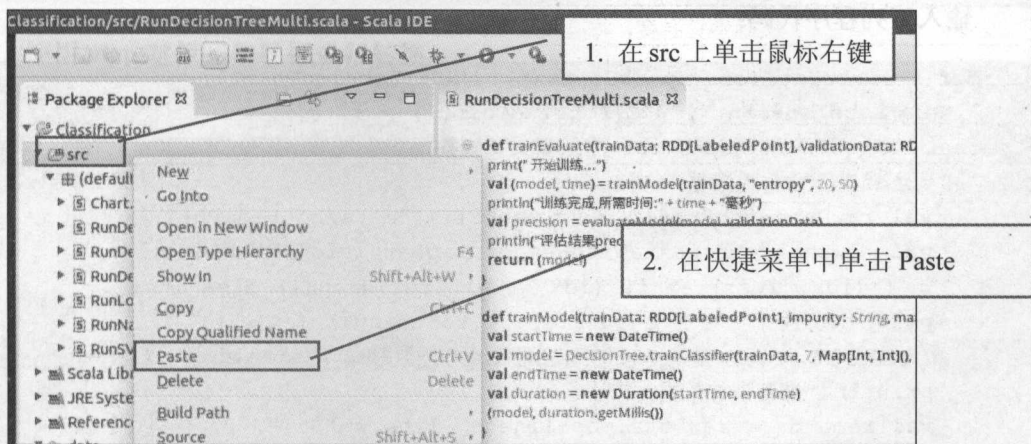


图 18-8 粘贴复制的程序

步骤 03 输入程序名称 (见图 18-9)

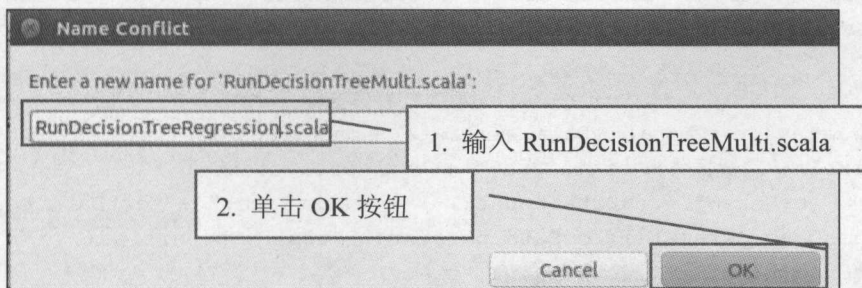


图 18-9 输入程序程序

步骤 04 修改 object name (见图 18-10)

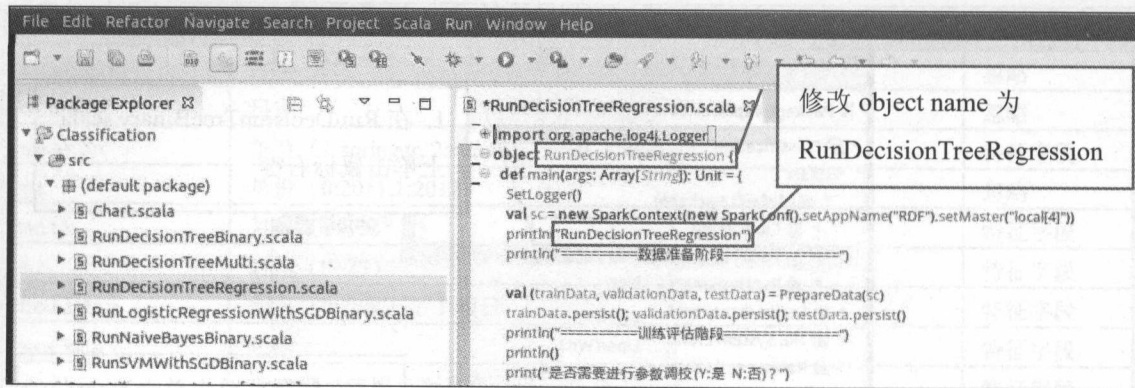


图 18-10 修改 object name

18.5 修改 RunDecisionTreeRegression.scala

步骤 01 修改 PrepareData()数据准备

输入下列程序代码:

```
def PrepareData(sc: SparkContext): (RDD[LabeledPoint],
  RDD[LabeledPoint], RDD[LabeledPoint]) = {
  //-----1.导入转换数据-----
  print("开始导入数据...")
  val rawDataWithHeader = sc.textFile("data/hour.csv")
  val rawData = rawDataWithHeader.mapPartitionsWithIndex
    { (idx, iter) => if (idx == 0) iter.drop(1) else iter }
  println("共计: " + rawData.count.toString() + "条")
  //-----2.创建训练评估所需数据 RDD[LabeledPoint]-----
  println("准备训练数据...")
  val records = rawData.map(line => line.split(","))
  val data = records.map { fields =>
    val label = fields(fields.size - 1).toInt
    val featureSeason = fields.slice(2, 3).map(d => d.toDouble)
    val features = fields.slice(4, fields.size - 3).map(d => d.toDouble)
    LabeledPoint(label, Vectors.dense(featureSeason ++ features))
  }
  //-----3.以随机方式将数据分为3个部分并且返回-----
  val Array(trainData, cvData, testData) = data.randomSplit(Array(0.8, 0.1, 0.1))
  println("将数据分为 trainData:" + trainData.count() +
    " cvData:" + cvData.count() + " testData:" + testData.count())
  trainData.persist(); cvData.persist(); testData.persist()
  return (trainData, cvData, testData)
}
```

上述程序代码的说明如下：

➤ 定义 PrepareData 函数

```
def PrepareData(sc: SparkContext): (RDD[LabeledPoint],
  RDD[LabeledPoint], RDD[LabeledPoint]) = {
```

PrepareData 函数传入参数 sc 是 SparkContext；返回：trainData、validationData、testData。

➤ 读取 hour.csv 并且进行预处理

```
val rawDataWithHeader = sc.textFile("data/hour.csv")
```

➤ 使用 sc.textFile 读取 hour.csv。

```
val rawData=rawDataWithHeader.mapPartitionsWithIndex
  { (idx, iter) => if (idx == 0) iter.drop(1) else iter }
```

第1行表头是字段名，所以先删除。

```
println("共计: " + rawData.count.toString() + "条")
```

输出数据记录数。

```
val records = rawData.map(line => line.split(","))
line.split(',')以“,”符号分隔字段，读取每一个字段。
```

➤ 使用 map 对每一条数据进行转换

```
val data = records.map { fields =>
...
}
```

使用 map 针对每一条数据进行转换，传入 fields 作为参数。

➤ 提取 label 标签字段

```
val label = fields(fields.size - 1).toInt
```

读取最后一个字段作为 label 字段。

➤ 提取 feature 特征字段

```
val featureSeason = fields.slice(2, 3) .map(d => d.toDouble)
```

读取季节字段作为 feature 特征字段，并且转换为 Double。

```
val features = fields.slice(4, fields.size - 3).map(d => d.toDouble)
```

读取其他字段（不含 casual 与 registered）作为 feature 特征字段，并且转换为 Double。

➤ 创建 LabeledPoint

```
LabeledPoint( label, Vectors.dense(featureSeason ++ features))
```


创建 LabelPoint, 作为后续训练数据。

- 标签字段: 由 featureSeason 与 features 数组相加。
- 特征字段: label。

➤ 随机方式分为 3 部分

```
val Array(trainData, cvData, testData) = data.randomSplit(Array(0.8, 0.1, 0.1))
data.randomSplit(Array(0.8, 0.1, 0.1))按照8:1:1比例, 以随机方式分为3个部分。
```

➤ 显示数据个数

```
println("将数据分为 trainData:" + trainData.count() +
      "    cvData:" + cvData.count() + "    testData:" + testData.count())
```

➤ 持久化数据

```
trainData.persist(); cvData.persist(); testData.persist()
```

持久化数据在内存中可以提高运行效率。

➤ 返回数据

```
return (trainData, cvData, testData)
```

步骤 02 修改 parametersTunning() 参数调校

在决策树回归分析中, 必须使用 DecisionTree.trainRegressor, 无 numClasses 参数, 而且 impurity 固定为 variance。

```
val model = DecisionTree.trainRegressor(trainData, Map[Int, Int](),
    impurity, maxDepth, maxBins)
```

因为 impurity 固定为 variance, 不须评估 Impurity 参数, 因此将 impurityArray 参数改为 Array("variance")。如图 18-11 所示。

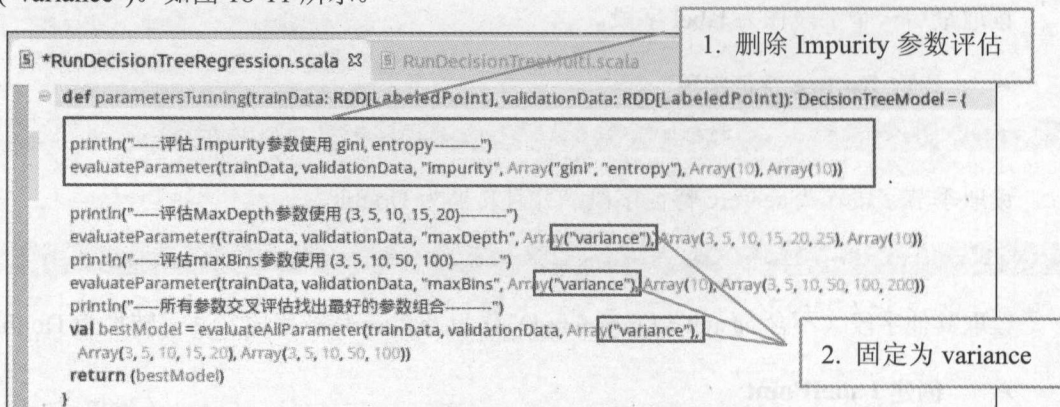


图 18-11 删除 Impurity 参数评估, 将 impurityArray 参数改为 Array("variance")

步骤 03 修改 trainModel 训练模型 (见图 18-12)

```

*RunDecisionTreeRegression.scala
def trainModel(trainData: RDD[LabeledPoint], impurity: String, maxDepth: Int, maxBins: Int): (DecisionTreeModel, Double) = {
  val startTime = new DateTime()
  val model = DecisionTree.trainRegressor(trainData, Map[Int, Int](), impurity, maxDepth, maxBins)
  val endTime = new DateTime()
  val duration = new Duration(startTime, endTime)
  (model, duration.getMillis())
}

```

修改为 DecisionTree.trainRegressor

图 18-12 修改 trainModel 训练模型

➤ 决策树二元或多元分类参数设置

在之前决策树二元或多元分类时, 我们使用 DecisionTree.trainClassifier。

```

val model = DecisionTree.trainClassifier(trainData, 2, Map[Int, Int](),
  impurity, maxDepth, maxBins)

```

➤ 决策树回归分析参数设置

但是在决策树回归分析时, 我们必须使用 DecisionTree.trainRegressor, 无 numClasses 参数, 而且 impurity 固定为 variance。

```

val model = DecisionTree.trainRegressor(trainData, Map[Int, Int](),
  impurity, maxDepth, maxBins)

```

步骤 04 修改 evaluateModel 评估模型计算 RMSE (见图 18-13)

```

*RunDecisionTreeRegression.scala
def evaluateModel(model: DecisionTreeModel, validationData: RDD[LabeledPoint]): (RMSE, Double) = {
  val scoreAndLabels = validationData.map { data =>
    val predict = model.predict(data.features)
    (predict, data.label)
  }
  val Metrics = new RegressionMetrics(scoreAndLabels)
  val RMSE = Metrics.rootMeanSquaredError
  (RMSE, Metrics.areaUnderROC)
}

```

改为 RegressionMetrics

图 18-13 修改 evaluateModel 评估模型计算 RMSE

➤ 决策树二元分类以 AUC 作为评估

在之前决策树二元分类 RunDecisionTreeBinary.scala 中, 我们是以 AUC 作为评估模型的准确率。所以我们使用 BinaryClassificationMetrics 先创建 Metrics, 然后使用.areaUnderROC 方法计算 AUC, 如下程序:

```

val Metrics = new BinaryClassificationMetrics(scoreAndLabels)
val AUC = Metrics.areaUnderROC
(AUC)

```

决策树回归分析以 RMSE 作为评估

在第 11.15 节中，我们创建了 `computeRMSE` 函数来计算。但是，其实 `MLlib` 本身也提供了计算 RMSE 的功能。我们可以使用 `RegressionMetrics` 先创建 `Metrics`，然后使用 `rootMeanSquaredError` 方法计算均方根差（RMSE），如下程序：

```
val Metrics = new RegressionMetrics(scoreAndLabels)
val RMSE = Metrics.rootMeanSquaredError
(RMSE)
```

步骤 05 修改 evaluateParameter 评估参数

将程序代码修改如下，改为 RMSE。RMSE 数值范围为 0,150。如图 18-14 所示。

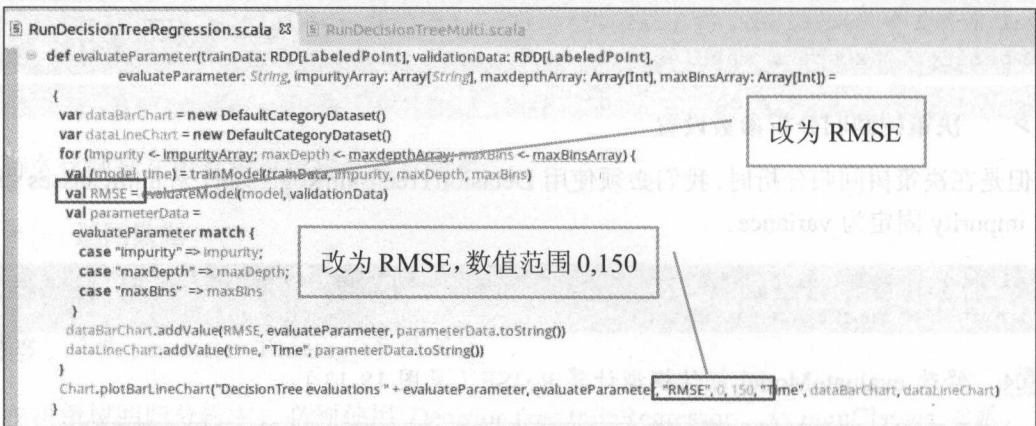


图 18-14 修改 evaluateParameter 评估参数

步骤 06 修改 evaluateAllParameter 参数交叉评估

`evaluateAllParameter` 程序将 3 个参数交叉评估，找出最好的参数组合。如图 18-15 所示。

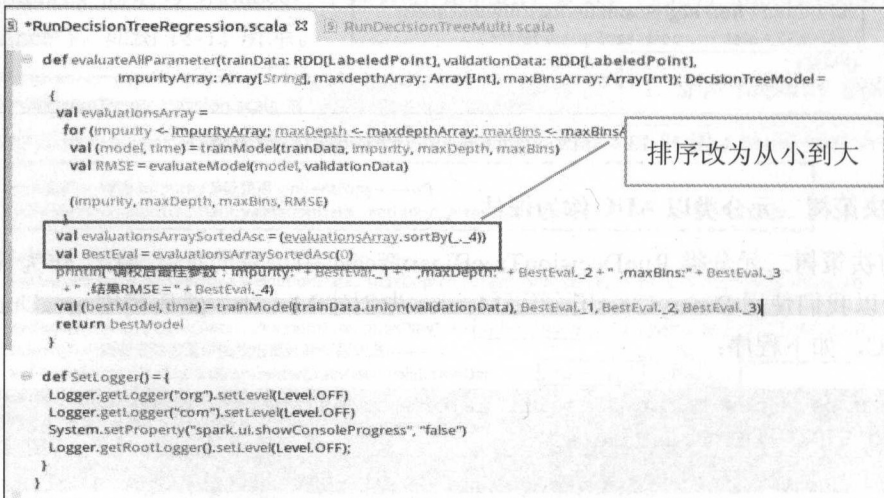


图 18-15 修改 evaluateAllParameter 参数交叉评估

➤ 决策树二元分类以 AUC 从大到小排序

在之前决策树二元分类 `RunDecisionTreeBinary.scala` 中，我们是以 AUC 作为评估模型的准确率，AUC 越大代表准确率越高。所以使用从大到小排序找出最佳的模型，如下列程序：

```
val BestEval = (evaluationsArray.sortBy(_._4).reverse)(0)
```

➤ 决策树回归分析以 RMSE 从小到大排序

但是在回归分析中，我们是以 RMSE 作为评估模型的准确率。RMSE 越小代表误差越小，准确率越高。所以使用从小到大排序，找出最佳的模型，如下列程序：

```
val evaluationsArraySortedAsc = (evaluationsArray.sortBy(_._4))
val BestEval = evaluationsArraySortedAsc(0)
```

步骤 07 修改 `PredictData()` 预测数据

输入下列程序代码：

```
def PredictData(sc: SparkContext, model: DecisionTreeModel): Unit = {
  //-----1.导入转换数据-----
  print("开始导入数据...")
  val rawDataWithHeader = sc.textFile("data/hour.csv")
  val rawData = rawDataWithHeader.mapPartitionsWithIndex
  { (idx, iter) => if (idx == 0) iter.drop(1) else iter }
  println("共计: " + rawData.count.toString() + "条")
  //-----2.创建训练评估所需数据 RDD[LabeledPoint]-----
  println("准备训练数据...")
  val records = rawData.map(line => line.split(","))
  val data = records.take(200).map { fields =>

    val label = fields(fields.size - 1).toInt
    val featureSeason = fields.slice(2, 3).map(d => d.toDouble)
    val features = fields.slice(4, fields.size - 3).map(d => d.toDouble)
    val featuresVectors = Vectors.dense(featureSeason ++ features)
    var dataDesc = ""
    dataDesc = dataDesc + { featuresVectors(0) match { case 1 => "春";
  case 2 => "夏"; case 3 => "秋"; case 4 => "冬"; } } + "天,"
    dataDesc = dataDesc + featuresVectors(1).toInt + "月,"
    dataDesc = dataDesc + featuresVectors(2).toInt + "时,"
    dataDesc = dataDesc + { featuresVectors(3) match
  { case 0 => "非假日"; case 1 => "假日"; } } + ","
    dataDesc = dataDesc + "星期" + { featuresVectors(4) match
  { case 0 => "日"; case 1 => "一"; case 2 => "二";
  case 3 => "三"; case 4 => "四"; case 5 => "五";
  case 6 => "六"; } } + ","
    dataDesc = dataDesc + { featuresVectors(5) match
  { case 1 => "工作日"; case 0 => "非工作日" } } + ","
```



```

    dataDesc = dataDesc + { featuresVectors(6) match{ case 1 => "晴";
case 2 => "阴"; case 3 => "小雨"; case 4 => "大雨" } } + ","
    dataDesc = dataDesc + (featuresVectors(7) * 41).toInt + "度,"
    dataDesc = dataDesc + "体感" + (featuresVectors(8) * 50).toInt + "度,"
    dataDesc = dataDesc + "湿度" + (featuresVectors(9) * 100).toInt + ","
    dataDesc = dataDesc + "风速" + (featuresVectors(10) * 67).toInt + ","

    val predict = model.predict(featuresVectors)
    val result = (if (label == predict) "正确" else "错误")
    val error = (math.abs(label - predict).toString())
    println(" 特征: " + dataDesc + " ==> 预测结果:" +
predict.toInt + "    实际:" + label.toInt + " 误差:" + error)
  }
}

```

PredictData 与 PrepareData 程序代码类似，以下仅说明不同之处。

➤ 定义 PredictData 函数

```
def PredictData( sc: SparkContext,model: DecisionTreeModel): Unit = {
```

PredictData 传入下列参数：

- sc: SparkContext
- model: 之前训练完成的模型

➤ 导入数据

```
val rawDataWithHeader = sc.textFile("data/hour.csv")
```

因为此数据集并没有提供预测的数据，所以我们仍然使用 hour.csv 进行预测。

➤ 随机数读取测试数据

```
val Array(pData, oData) = rawData.randomSplit(Array(0.1, 0.9))
```

➤ 读取每一个字段

```
val records = pData.map(line => line.split(","))
```

以","分隔字段，读取每一个字段。

➤ 读取 20 条数据进行预测

```
val data = records.take(20).map { fields =>
...
}
```

我们在这里只读取 20 条数据进行预测。

➤ 提取 label 标签字段

```
val label = fields(fields.size - 1).toInt
```

读取最后一个字段作为 label 字段，用于后续对比实际与预测数据时判断是否正确。

➤ 创建 featuresVectors

```
val featureSeason = fields.slice(2, 3) .map(d => d.toDouble)
```

读取季节字段作为 feature 特征字段，并且转换为 Double。

```
val features = fields.slice(4, fields.size - 3).map(d => d.toDouble)
```

读取其他字段不含 casual 与 registered 作为 feature 特征字段，并且转换为 Double。

```
val featuresVectors = Vectors.dense(featureSeason ++ features)
```

创建 featuresVectors，由 featureSeason 与 features 数组相加，作为后续的预测数据。

➤ 定义 dataDesc

```
var dataDesc = ""
```

因为特征字段都是数字，对用户没有意义。因此我们先定义 dataDesc 变量，后续将特征字段转换为文字描述。

➤ 显示季节

```
dataDesc = dataDesc + { featuresVectors(0) match { case 1 => "春";  
    case 2 => "夏"; case 3 => "秋"; case 4 => "冬"; } } + "天,"
```

显示季节。原本是 1,2,3,4，转换后为春、夏、秋、冬。

➤ 显示月份

```
dataDesc = dataDesc + featuresVectors(1).toInt + "月,"
```

➤ 显示假日或非假日

```
dataDesc = dataDesc + { featuresVectors(3)  
    match { case 0 => "非假日"; case 1 => "假日"; } } + ","
```

➤ 显示星期

```
dataDesc = dataDesc + "星期" + { featuresVectors(4) match {  
    case 0 => "日"; case 1 => "一"; case 2 => "二"; case 3 => "三";  
    case 4 => "四"; case 5 => "五"; case 6 => "六"; } } + ","
```

➤ 显示工作日 / 非工作日

```
dataDesc = dataDesc + { featuresVectors(5) match  
    { case 1 => "工作日"; case 0 => "非工作日" } } + ","
```

➤ 显示天气

```
dataDesc = dataDesc + { featuresVectors(6) match{ case 1 => "晴";  
case 2 => "阴"; case 3 => "小雨"; case 4 => "大雨" } } + ", "
```

➤ 显示气温

```
dataDesc = dataDesc + (featuresVectors(7) * 41).toInt + "度, "
```

显示的气温是摄氏温度。因为原始数据已经是将温度除以 41 进行了标准化，所以再乘以 41 还原为真实的温度。

➤ 显示体感温度

```
dataDesc = dataDesc + "体感" + (featuresVectors(8) * 50).toInt + "度, "
```

实际感觉温度因为原始数据已经是将感觉温度除以 50 进行了标准化，所以再乘以 50 还原为真实的温度。

➤ 显示湿度

```
dataDesc = dataDesc + "湿度" + (featuresVectors(9) * 100).toInt + ", "
```

因为原始数据已经是将湿度除以 100 进行了标准化，所以再乘以 100 还原为真实的湿度。

➤ 显示风速

```
dataDesc = dataDesc + "风速" + (featuresVectors(10) * 67).toInt + ", "
```

因为原始数据已经是将风速除以 67 进行了标准化，所以再乘以 67 还原为真实的风速。

➤ 进行预测

```
val predict = model.predict(featureVector)
```

使用 model.predict 传入 featureVector 进行预测。

➤ 判断是否正确

```
val result = (if (label == predict) "正确" else "错误")
```

对比实际与预测数据，判断是否正确。

➤ 计算误差

```
val error = (math.abs(label - predict).toString())
```

实际与预测相减，计算误差

➤ 打印预测结果

```
println(" 特征: " + dataDesc + " ==> 预测结果: " +  
predict.toInt + " 实际: " + label.toInt + " 误差: " + error)
```

18.6

运行 RunDecisionTreeRegression.
scala 进行参数调校

步骤 01 依次选择菜单及其菜单项 Run→Run Configurations (见图 18-16)

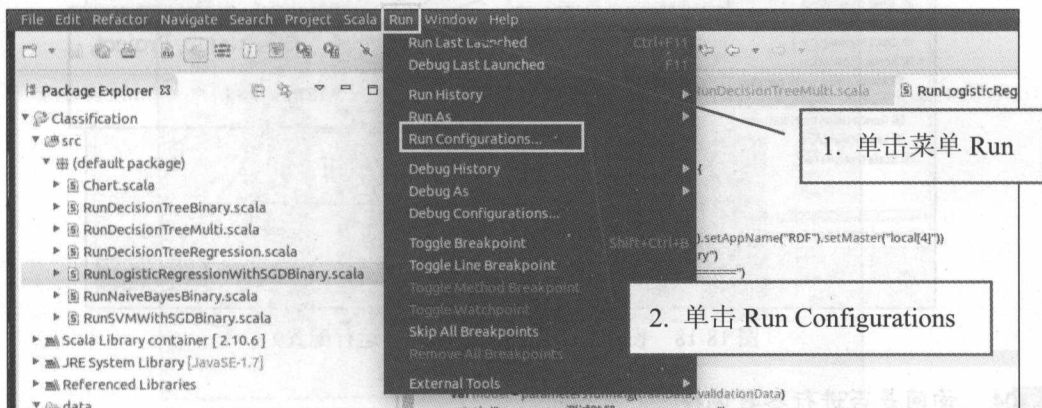


图 18-16 依次选择菜单及其菜单项 Run→Run Configurations

步骤 02 复制 Run Configurations (见图 18-17)

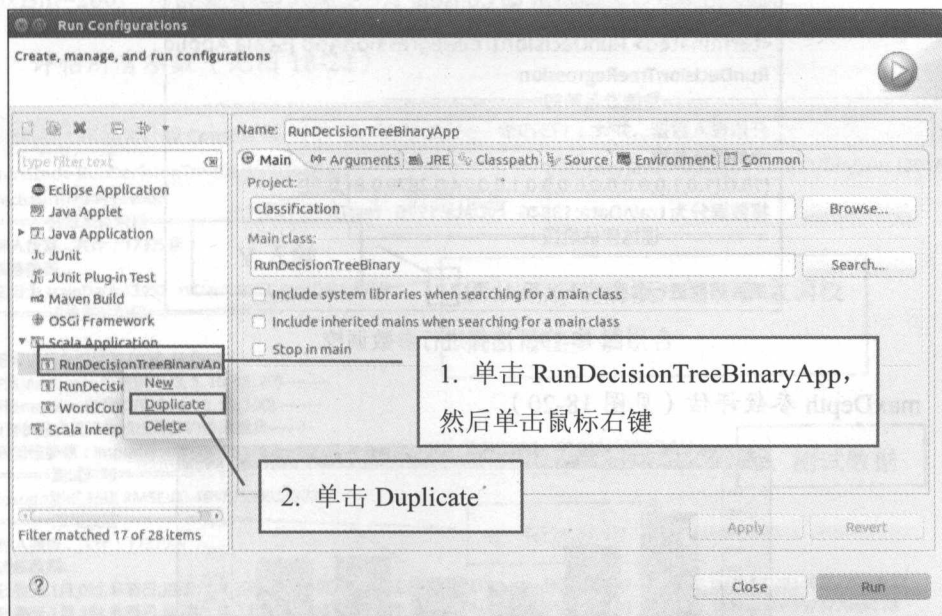


图 18-17 复制 Run Configurations (运行配置)

步骤 03 设置 Run Configurations (见图 18-18)

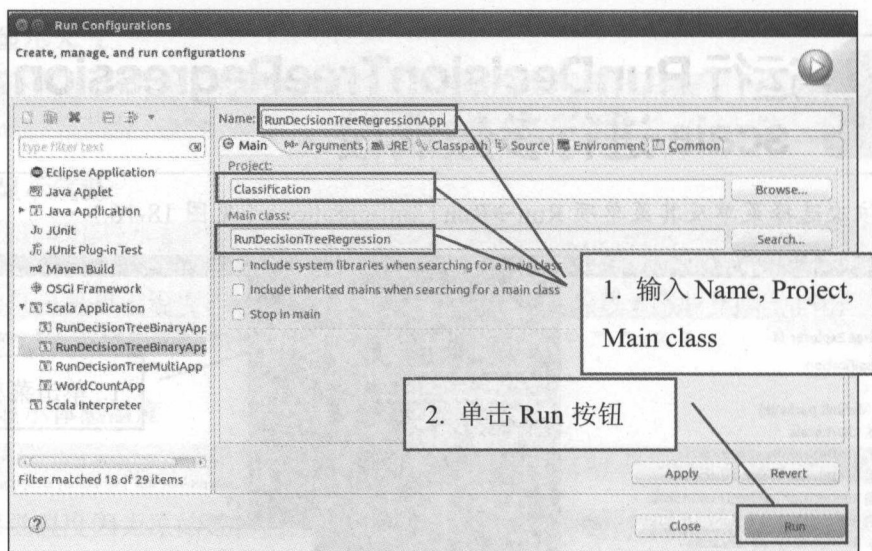


图 18-18 设置 Run Configurations (运行配置)

步骤 04 询问是否进行参数调校

程序会询问是否需要进行参数调校 (Y: 是、N: 否)? 输入 Y。如图 18-19 所示。



图 18-19 选择进行参数调校

步骤 05 maxDepth 参数评估 (见图 18-20)

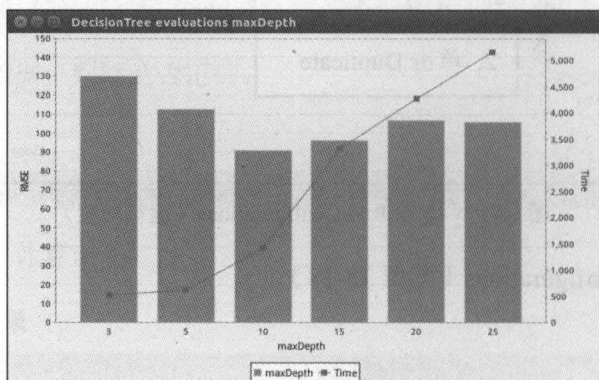


图 18-20 maxDepth 参数评估的结果图

从图 18-20 可以看到 $\text{maxDepth}=10$ 时, RMSE 最低。随着 maxDepth 越大, 所需的时间越多。所以 $\text{maxDepth}=10$, 可能是不错的选择。

步骤 06 maxBins 参数评估 (见图 18-21)

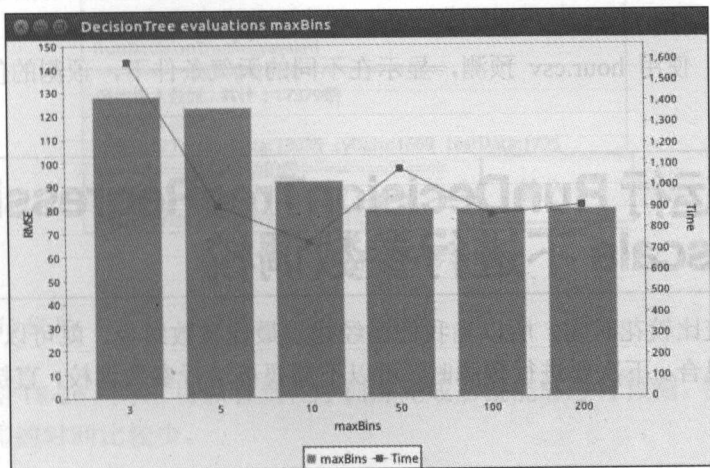


图 18-21 maxBins 参数评估的结果图

从图 18-21 可以看到 $\text{maxBin}=200$ 时, RMSE 最低。随着 maxBin 越大, 所需的时间越多。所以 $\text{maxBin}=200$, 可能是不错的选择。

步骤 07 评估所有参数 (见图 18-22)

Problems Tasks Console
/lib/jvm/java-7-openjdk-amd64/bin/java (2016年5月19)

```

<terminated> RunDecisionTreeRegressionApp
RunDecisionTreeRegression
=====数据准备阶段=====
开始导入数据... 共计: 17379条
准备训练数据...
将数据分为 trainData:13952 cvData:1669 testData:1758
=====训练评估阶段=====

是否需要参数调校(Y:是 N:否)? Y
--评估MaxDepth参数使用(3, 5, 10, 15, 20)-----
--评估maxBins参数使用(3, 5, 10, 50, 100)-----
--所有参数交叉评估找出最好的参数组合-----
调校后最佳参数: impurity:variance ,maxDepth:10 ,maxBins:100 ,结果RMSE = 79.9151166950444
=====测试阶段=====
使用testata测试, 结果 RMSE:80.40373528053972
=====预测数据=====
开始导入数据... 共计: 17379条
准备训练数据...
特征: 春天,1月,0时,非假日,星期六,非工作日,晴,9度,体感14度,湿度81,风速0, ==> 预测结果:33 实际:16 误差:17.393442622950822
特征: 春天,1月,1时,非假日,星期六,非工作日,晴,9度,体感13度,湿度80,风速0, ==> 预测结果:33 实际:40 误差:6.606557377049178
特征: 春天,1月,2时,非假日,星期六,非工作日,晴,9度,体感13度,湿度80,风速0, ==> 预测结果:25 实际:32 误差:6.524999999999999
特征: 春天,1月,3时,非假日,星期六,非工作日,晴,9度,体感14度,湿度75,风速0, ==> 预测结果:14 实际:13 误差:1.3571428571428577
特征: 春天,1月,4时,非假日,星期六,非工作日,晴,9度,体感14度,湿度75,风速0, ==> 预测结果:4 实际:1 误差:3.584905660377358
特征: 春天,1月,5时,非假日,星期六,非工作日,阴,9度,体感12度,湿度75,风速6, ==> 预测结果:4 实际:1 误差:3.584905660377358
特征: 春天,1月,6时,非假日,星期六,非工作日,晴,9度,体感13度,湿度80,风速0, ==> 预测结果:6 实际:2 误差:4.166666666666667
特征: 春天,1月,7时,非假日,星期六,非工作日,晴,8度,体感12度,湿度86,风速0, ==> 预测结果:13 实际:3 误差:10.724137931034482
特征: 春天,1月,8时,非假日,星期六,非工作日,晴,9度,体感14度,湿度75,风速0, ==> 预测结果:46 实际:8 误差:38.36363636363637
特征: 春天,1月,9时,非假日,星期六,非工作日,晴,13度,体感17度,湿度76,风速0, ==> 预测结果:140 实际:14 误差:126.0810810810811

```

1. 数据准备

2. 训练评估阶段, 进行参数调校
找出最佳参数组合

3. 测试数据

4. 预测数据

图 18-22 评估所有参数

1. **数据准备阶段**: 显示导入个数以及 trainData、validationData、testData 个数。
2. **训练评估阶段**: 进行参数调校, 找出最佳参数组合。调校后最佳参数: impurity:variance, maxDepth:10, maxBins:100, 结果 RMSE = 79.9。
3. **测试阶段**: 使用测试数据再测试模型, RMSE 大约 80.4。与训练阶段差异不大, 确认没有 overfitting 的问题。
4. **预测阶段**: 使用 hour.csv 预测, 显示在不同的天气条件下, 预测的值与实际的值, 并显示误差。

18.7

运行 RunDecisionTreeRegression.
scala 不进行参数调校

因为参数调校比较花时间, 所以当我们已经找出最佳参数组合, 就可以修改 trainEvaluate 程序为最佳参数组合。下次要进行预测时, 可以不需要再运行参数调校, 直接使用最佳参数进行预测。

步骤 01 修改 trainEvaluate 为最佳参数组合

修改 trainEvaluate 程序, 改为最佳参数组合。如图 18-23 所示。

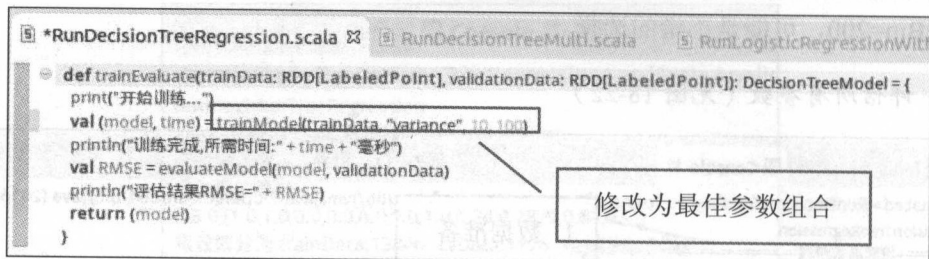


图 18-23 修改 trainEvaluate 为最佳参数组合

步骤 02 开始运行程序 (见图 18-24)

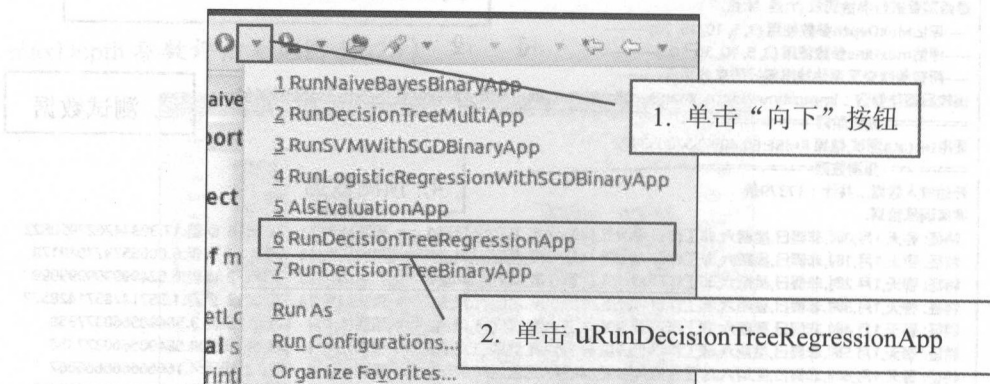


图 18-24 开始运行 RunDecisionTreeRegressionApp 程序

步骤 03 是否需要进行参数调校

再次运行程序，程序会询问是否需要进行参数调校（Y：是、N：否）？输入 N。如图 18-25 所示。



图 18-25 选择不进行参数调校

步骤 04 再次运行程序

运行结果如图 18-26 所示，可以看到程序训练完成后就能够进行预测。因为不进行参数调校，所以运行所需的时间比较少。

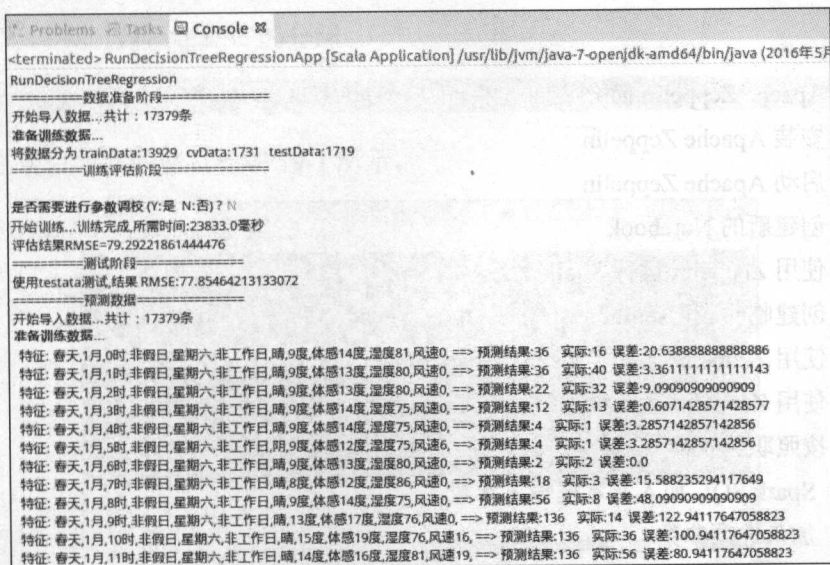


图 18-26 选择最佳参数组合且不再进行参数调校之后的运行结果

第 19 章

使用Apache Zeppelin 数据可视化

- 19.1 Apache Zeppelin 简介
- 19.2 安装 Apache Zeppelin
- 19.3 启动 Apache Zeppelin
- 19.4 创建新的 Notebook
- 19.5 使用 Zeppelin 运行 Shell 命令
- 19.6 创建临时表 UserTable
- 19.7 使用 Zeppelin 运行年龄统计 Spark SQL
- 19.8 使用 Zeppelin 运行性别统计 Spark SQL
- 19.9 按照职业统计
- 19.10 Spark SQL 加入文本框输入参数
- 19.11 加入选项参数
- 19.12 同时显示多个统计字段
- 19.13 设置工具栏
- 19.14 设置段落标题
- 19.15 设置 Paragraph 段落的宽度
- 19.16 设置显示模式

Zeppelin 由 NFLAB 开发提供了 Web 界面, 类似 iPython 的 Notebook, 可用于数据分析与数据可视化。它支持 Apache Spark Scala、Apache Spark Python、SparkSQL、Hive、Markdown and Shell。Zeppelin Web 界面与之前章节所介绍的 Spark-shell 相同, 都可以输入 Spark 命令并且立刻得到响应, 但是它比 Spark-shell 提供了更多功能。

➤ Apache Zeppelin 使用命令的整理

本章使用的命令已经整理在本书的博客文章中, 在练习安装时可以复制在博客文章中的命令, 然后粘贴到终端程序中。这样可节省你打字的时间, 也不用担心打错字 (如果你无法在 VirtualBox 虚拟机的 Ubuntu “终端” 程序中执行复制/粘贴操作时, 参考第 3.9 节的说明, 设置好 VirtualBox 的共享剪贴板)。本书博客的网址为:

<http://blog.sina.com.cn/hadoopsparkbook>

19.1 Apache Zeppelin 简介

可以在下列网址查看 Apache 的介绍:

<https://zeppelin.incubator.apache.org/>

从该网址可以看到的网页如图 19-1 所示。

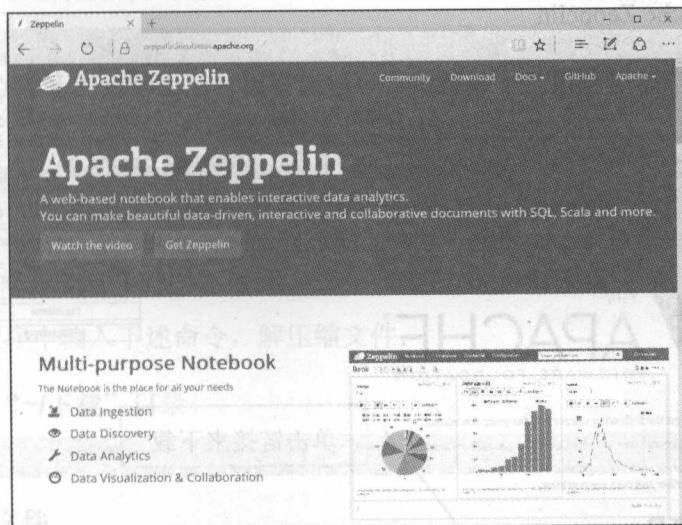


图 19-1 Apache Zeppelin 的官网页面

19.2 安装 Apache Zeppelin

步骤 01 下载 Apache Zeppelin 的页面

首先，在浏览器输入下列网址以便下载 Apache Zeppelin：

<https://zeppelin.incubator.apache.org/download>

可以下载已经编译完成的最新版本，如图 19-2 所示。

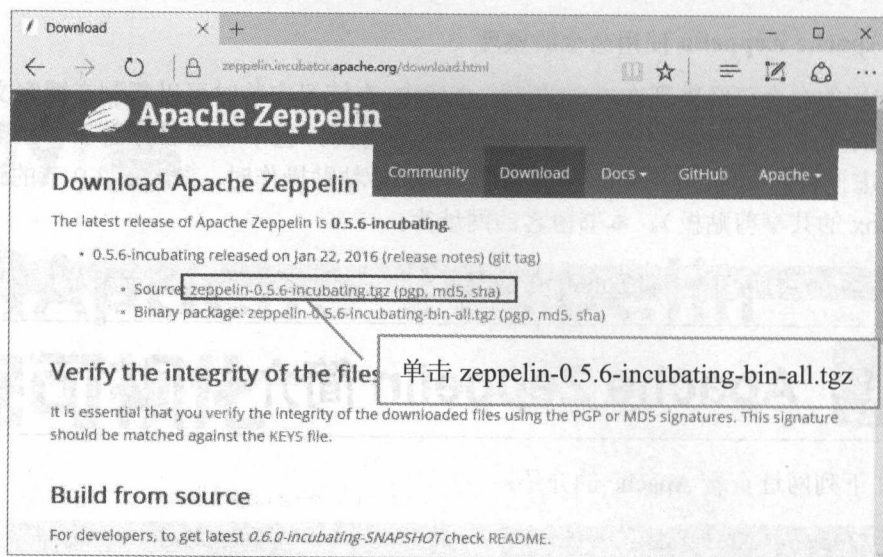


图 19-2 Apache Zeppelin 官网的下载页面

步骤 02 下载 Apache Zeppelin

进入下载页面，这里有很多下载链接，可以选择离自己比较近的下载点（镜像网站）。如图 19-3 所示。

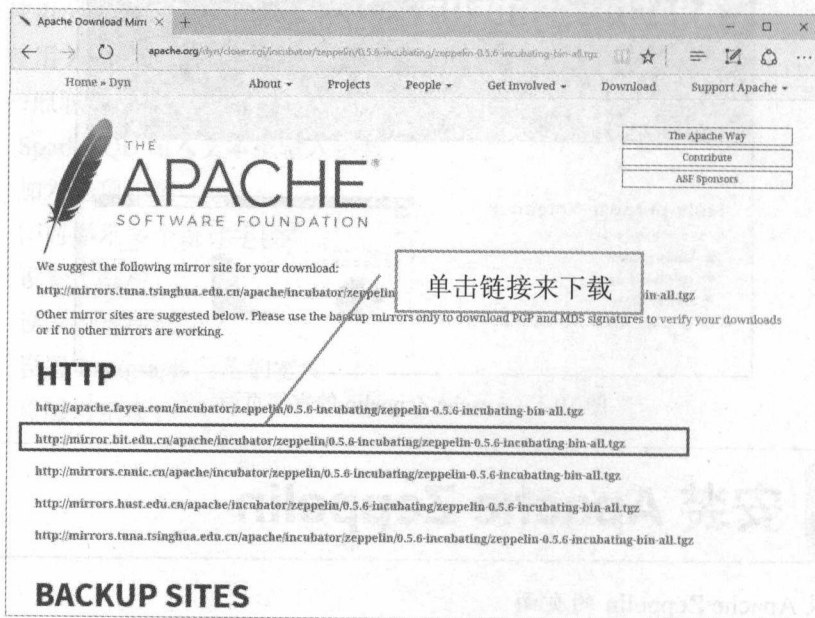


图 19-3 可以选择离自己比较近的下载点（镜像网站）来下载

步骤 03 保存文件（见图 19-4）

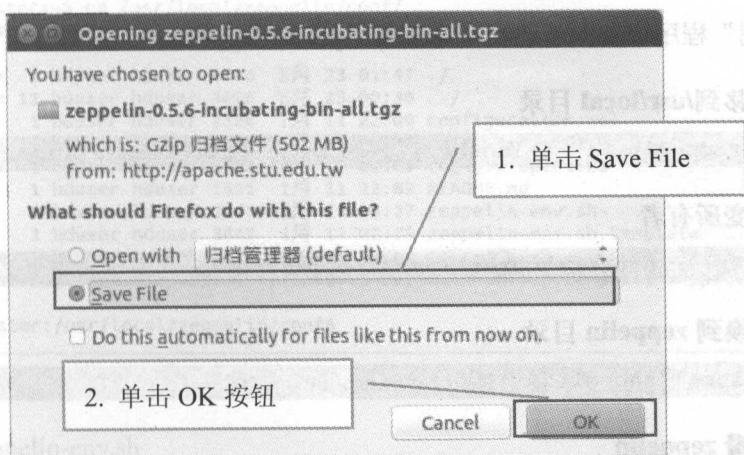


图 19-4 保存下载的文件

步骤 04 查看下载文件

首先，在终端程序中输入下述命令查看下载的 Apache Zeppelin：

```
11 ~/下载
```

可以看到已经下载的文件，如图 19-5 所示。

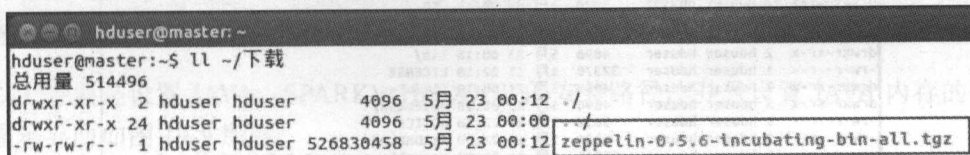


图 19-5 查看下载的 Apache Zeppelin

步骤 05 解压缩文件

在“终端”程序中输入下述命令，解压缩文件：

➤ 切换到“~/下载”目录

```
cd ~/下载
```

➤ 解压缩文件

```
tar zxvf zeppelin-0.5.6-incubating-bin-all.tgz
```

运行后屏幕显示界面如图 19-6 所示。

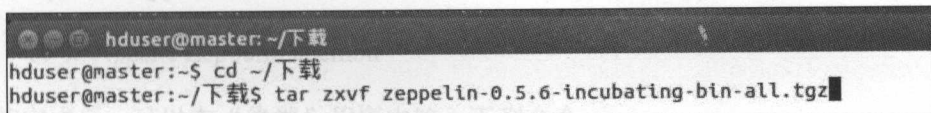


图 19-6 解压缩下载的 Apache Zeppelin 文件

步骤 06 安装 Apache Zeppelin

在“终端”程序中输入下述命令来安装 Apache Zeppelin:

➤ 移动到/usr/local 目录

```
sudo mv zeppelin-0.5.6-incubating-bin-all /usr/local/zeppelin
```

➤ 改变所有者

```
sudo chown -R hduser:hduser /usr/local/zeppelin
```

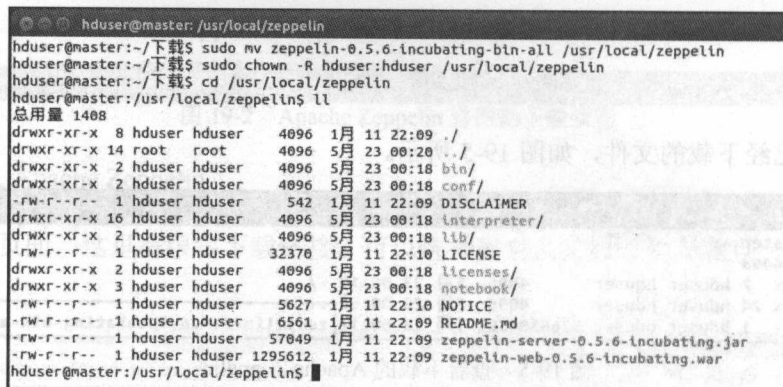
➤ 切换到 zeppelin 目录

```
cd /usr/local/zeppelin
```

➤ 查看 zeppelin

```
ll
```

运行后屏幕显示界面如图 19-7 所示。



```
hduser@master: /usr/local/zeppelin
hduser@master:~/下载$ sudo mv zeppelin-0.5.6-incubating-bin-all /usr/local/zeppelin
hduser@master:~/下载$ sudo chown -R hduser:hduser /usr/local/zeppelin
hduser@master:~/下载$ cd /usr/local/zeppelin
hduser@master: /usr/local/zeppelin$ ll
总用量 1408
drwxr-xr-x  8 hduser hduser  4096 1月 11 22:09 ./
drwxr-xr-x 14 root   root    4096 5月 23 00:20 ../
drwxr-xr-x  2 hduser hduser  4096 5月 23 00:18 bin/
drwxr-xr-x  2 hduser hduser  4096 5月 23 00:18 conf/
-rw-r--r--  1 hduser hduser   542 1月 11 22:09 DISCLAIMER
drwxr-xr-x 16 hduser hduser  4096 5月 23 00:18 interpreter/
drwxr-xr-x  2 hduser hduser  4096 5月 23 00:18 lib/
-rw-r--r--  1 hduser hduser 32370 1月 11 22:10 LICENSE
drwxr-xr-x  2 hduser hduser  4096 5月 23 00:18 licenses/
drwxr-xr-x  3 hduser hduser  4096 5月 23 00:18 notebook/
-rw-r--r--  1 hduser hduser   5627 1月 11 22:10 NOTICE
-rw-r--r--  1 hduser hduser   5661 1月 11 22:09 README.md
-rw-r--r--  1 hduser hduser   57049 1月 11 22:09 zeppelin-server-0.5.6-incubating.jar
-rw-r--r--  1 hduser hduser 1295612 1月 11 22:09 zeppelin-web-0.5.6-incubating.war
hduser@master: /usr/local/zeppelin$
```

图 19-7 安装 Apache Zeppelin

步骤 07 设置 Apache Zeppelin

Apache Zeppelin 的配置文件都存储在 conf 目录中,其中最主要的是 zeppelin-env.sh。在 conf 目录有 zeppelin-env.sh.template 模板文件,可以先复制模板文件再修改。在“终端”程序中输入下述命令:

```
cd /usr/local/zeppelin/conf/
ll
cp zeppelin-env.sh.template zeppelin-env.sh
```

运行后屏幕显示界面如图 19-8 所示。

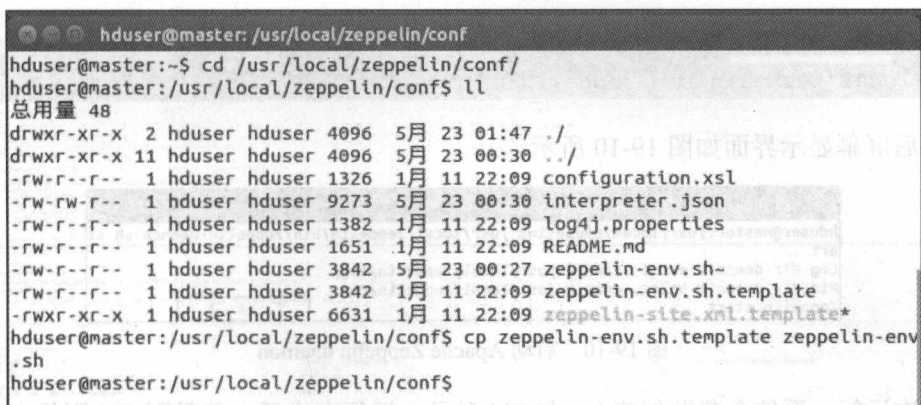


图 19-8 复制 zeppelin-env.sh.template 模板文件

步骤 08 编辑 zeppelin-env.sh

安装完成后，可以在“终端”程序中输入下列命令，编辑 zeppelin-env.sh：

```
sudo gedit /usr/local/zeppelin/conf/zeppelin-env.sh
```

启动 gedit 后，输入下列内容

```

export JAVA_HOME=/usr/lib/jvm/Java-7-openjdk-amd64
export SPARK_HOME=/usr/local/spark
export HADOOP_CONF_DIR=/usr/local/hadoop/etc/hadoop
export ZEPPELIN_INTP_JAVA_OPTS="-XX:PermSize=512M -XX:MaxPermSize=1024M"

```

以上主要是设置 JAVA、SPARK、HADOOP 的安装路径，并且设置 JAVA 内存的使用量，屏幕显示界面如图 19-9 所示。

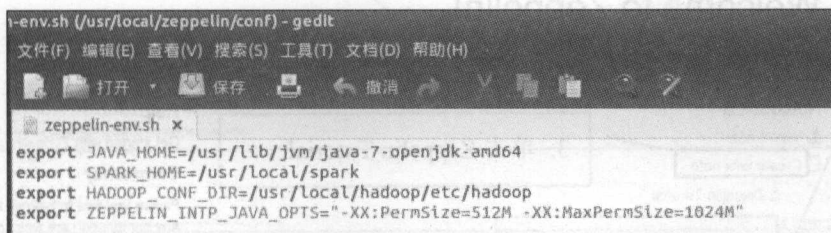


图 19-9 编辑 zeppelin-env.sh

19.3 启动 Apache Zeppelin

以上 Apache Zeppelin 已经安装完成，现在可以开始启动 Zeppelin 了。

步骤 01 启动 Apache Zeppelin daemon

安装完成后，可以在“终端”程序中输入下列命令：

➤ 启动 Apache Zeppelin daemon

```
/usr/local/zeppelin/bin/zeppelin-daemon.sh start
```

运行后屏幕显示界面如图 19-10 所示。

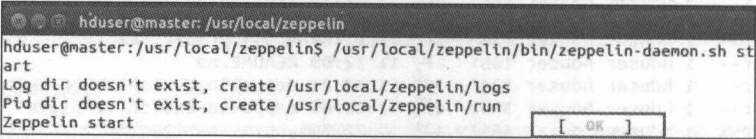


图 19-10 启动 Apache Zeppelin daemon

第一次运行，系统会帮助创建 log 与 Pid 目录。运行完成后，若看到 OK 则代表已经启动成功。

步骤 02 打开 Apache Zeppelin Web UI 界面

启动 Apache Zeppelin daemon 后，可以在 master 虚拟机的浏览器使用 Apache Zeppelin，链接到下列网址：

➤ 打开 Apache Zeppelin Web UI 界面（见图 19-11）

```
http://master:8080/
```

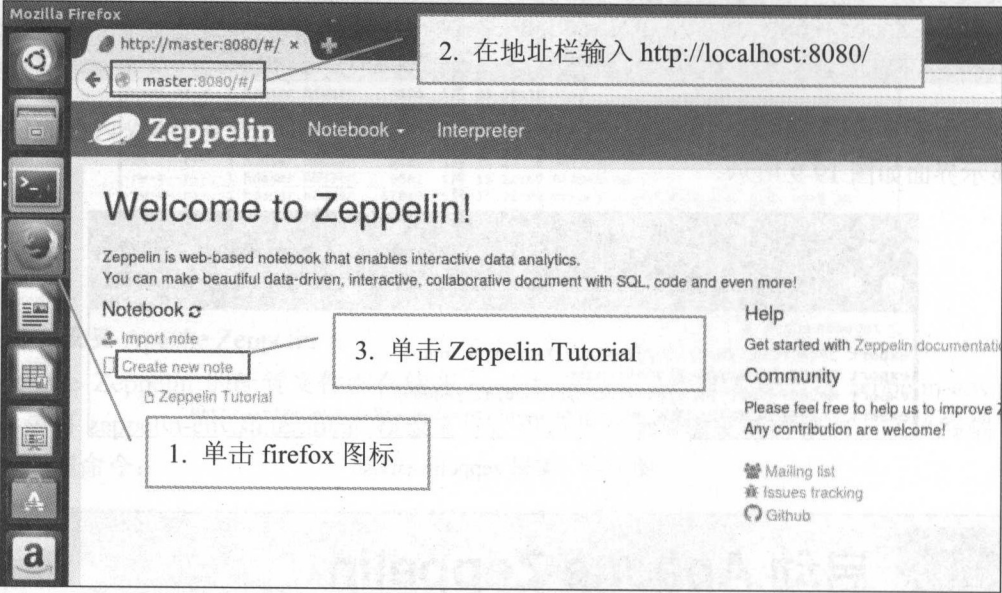


图 19-11 打开 Apache Zeppelin Web UI 界面

步骤 03 Zeppelin Tutorial Notebook

单击 Zeppelin Tutorial 后，会显示出 Zeppelin Tutorial Notebook 现成的范例供用户参考。如图 19-12 所示。

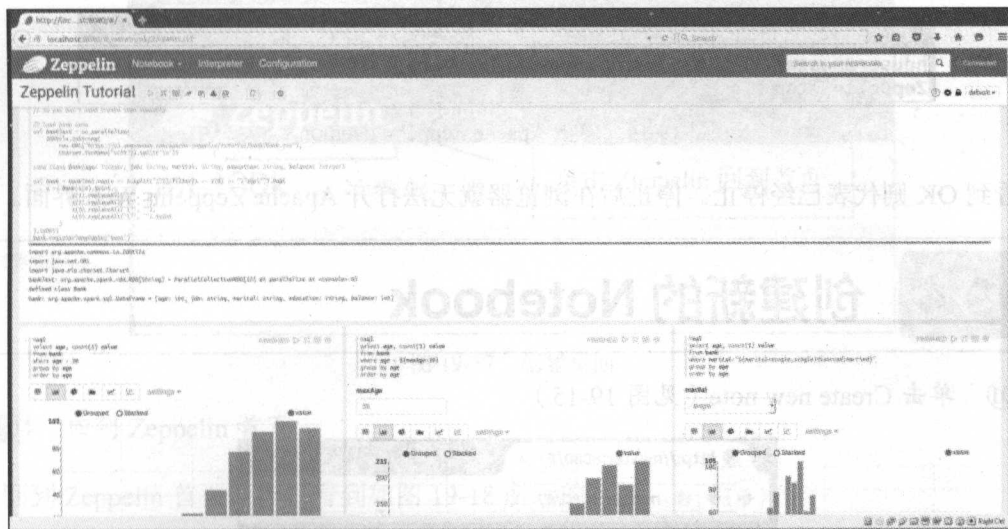


图 19-12 打开 Apache Zeppelin Web UI 界面

步骤 04 段落 (paragraph)

在 Zeppelin Tutorial Notebook 中可以看到 Notebook 内，能够创建多个段落 (Paragraph)，每个段落都可以输入不同的命令并且运行。如图 19-13 所示。

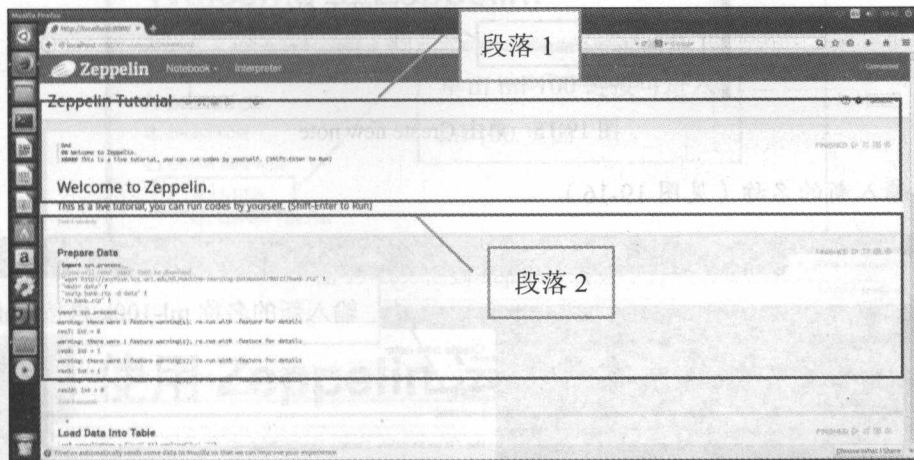


图 19-13 段落

步骤 05 停止 Apache Zeppelin daemon

如果不想继续使用 Apache Zeppelin，可以在“终端”程序中输入下列命令：

➤ 停止 Apache Zeppelin daemon

```
/usr/local/zeppelin/bin/zeppelin-daemon.sh stop
```

运行后屏幕显示界面如图 19-14 所示。

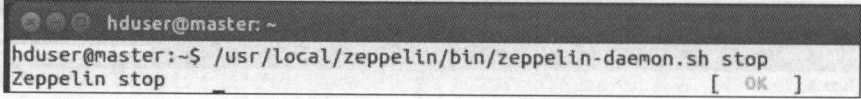


图 19-14 停止 Apache Zeppelin daemon

看到 OK 则代表已经停止。停止后在浏览器就无法打开 Apache Zeppelin Web 界面。

19.4 创建新的 Notebook

步骤 01 单击 Create new note (见图 19-15)



图 19-15 单击 Create new note

步骤 02 输入新的名称 (见图 19-16)

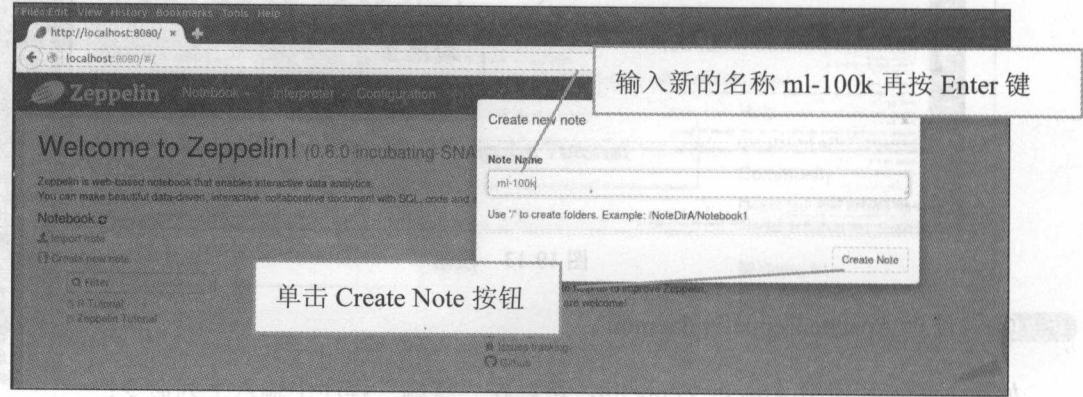


图 19-16 输入新的名称

步骤 03 创建页面

创建完成后就可以看到 ml-100k 页面，单击 Zeppelin 可以回到首页。如图 19-17 所示。

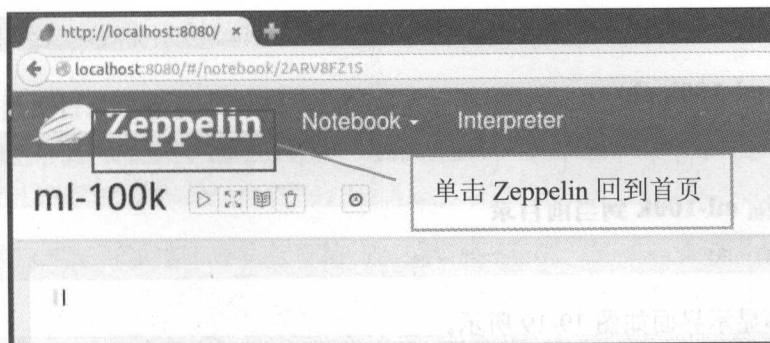


图 19-17 创建页面

步骤 04 回到 Zeppelin 首页

回到 Zeppelin 首页，可以看到如图 19-18 所示的页面。



图 19-18 回到 Zeppelin 首页

19.5 使用 Zeppelin 运行 Shell 命令

接下来，我们将以 ml-100k 数据集（可参考第 11.4 节的介绍），示范如何使用 Spark SQL 进行数据分析与数据可视化。

步骤 01 下载并解压缩数据文件

在“终端”程序中输入下列命令，下载并解压缩数据文件：

➤ 创建用于存储下载数据的目录

```
mkdir -p ~/workspace/zeppelin/data
```

➤ 切换到存储目录

```
cd ~/workspace/zeppelin/data
```

➤ 下载 ml-100k.zip

```
wget http://files.grouplens.org/datasets/movielens/ml-100k.zip
```

➤ 解压缩 ml-100k 到当前目录

```
unzip -j ml-100k
```

运行后屏幕显示界面如图 19-19 所示。

```

hduser@master: ~/workspace/zeppelin/data
hduser@master:~$
hduser@master:~$ mkdir -p ~/workspace/zeppelin/data
hduser@master:~$ cd ~/workspace/zeppelin/data
hduser@master:~/workspace/zeppelin/data$ wget http://files.grouplens.org/datasets/movielens/ml-100k.zip
--2016-05-19 13:04:08-- http://files.grouplens.org/datasets/movielens/ml-100k.zip
正在解析主机 files.grouplens.org (files.grouplens.org)... 128.101.34.146
正在连接 files.grouplens.org (files.grouplens.org)[128.101.34.146]:80... 已连接
已发出 HTTP 请求，正在等待回应... 200 OK
长度：4924029 (4.7M) [application/zip]
正在保存至：“ml-100k.zip”
100%[=====] 4,924,029 1.26MB/s 用时 4.6s
2016-05-19 13:04:13 (1.03 MB/s) - 已保存 “ml-100k.zip” [4924029/4924029]
hduser@master:~/workspace/zeppelin/data$ unzip -j ml-100k
Archive: ml-100k.zip
  inflating: allbut.pl
  inflating: mku.sh
  inflating: README
  inflating: u.data
  inflating: u.genre
  inflating: u.info
  inflating: u.item

```

图 19-19 下载并解压缩数据文件

步骤 02 Notebook 段落

在 ml-100k Notebook 新的段落中输入命令，屏幕显示界面如图 19-20 所示：

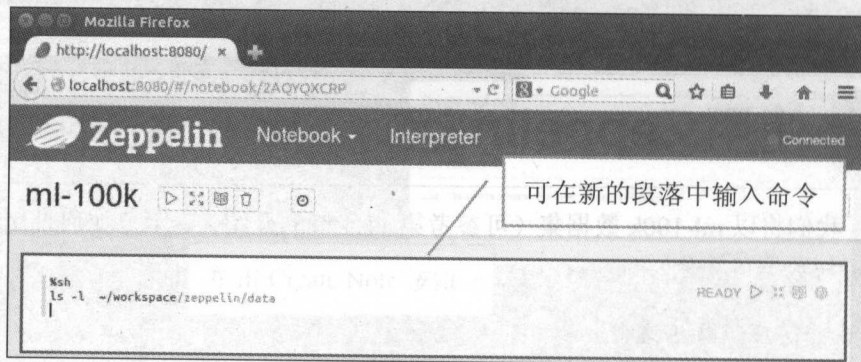


图 19-20 Notebook 段落

步骤 03 列出 ml-100k 文件列表

Zeppelin 支持 shell 命令，就好像在终端输入命令一样。要输入 shell 命令时，先输入 %sh。按 Enter 键后再输入命令。%sh 主要功能是告诉 Zeppelin 的解释器 (Interpreter)，后续要输入的是 shell 命令。

在新段落中输入下列命令：

➤ 列出数据目录文件

```
%sh
ls -l ~/workspace/zeppelin/data
```

执行命令的方式：可以单击“执行”按钮；或者按 Shift + Enter 组合键。屏幕显示界面如图 19-21 所示。

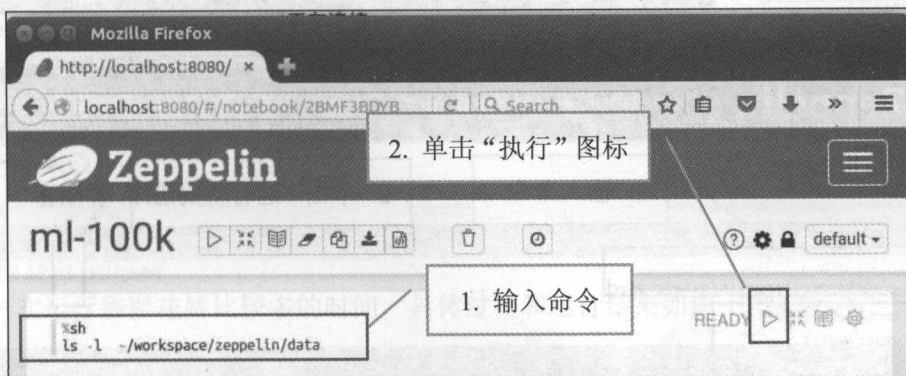


图 19-21 列出 ml-100k 文件列表的命令

单击 Ready 之后的“运行”按钮后，就会显示如图 19-22 所示的执行结果，即列出 ml-100k 的文件列表。

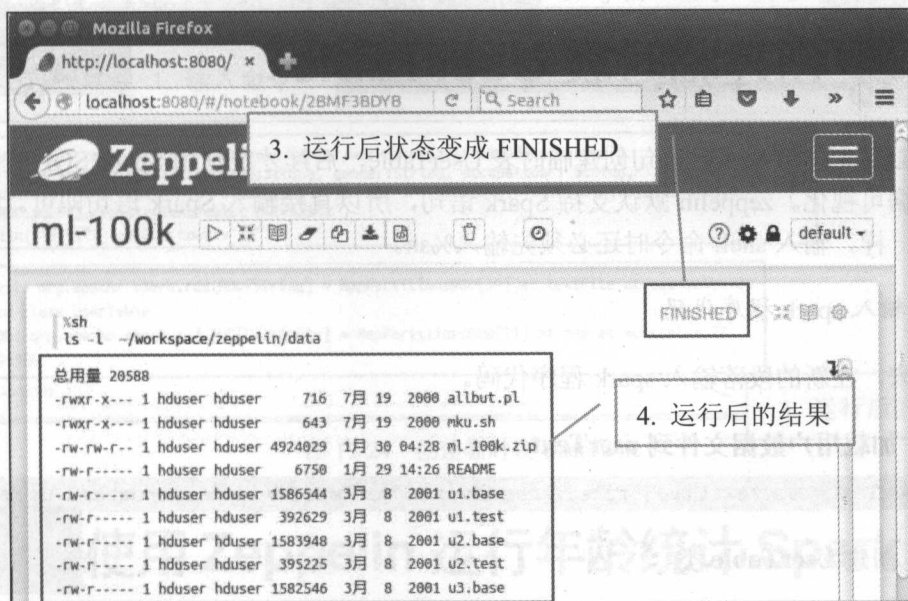


图 19-22 列出 ml-100k 文件列表的命令的运行结果

步骤 04 查看 u.user

我们也可以使用下列命令：

➤ 查看 u.user 的数据前几条数据

```
%sh
head ~/workspace/zeppelin/data/u.user
```

屏幕显示界面如图 19-23 所示，运行后会显示 u.user 数据。字段是序号、年龄、职业、邮政编码。我们后续会以这个文件示范 Zeppelin 的功能。

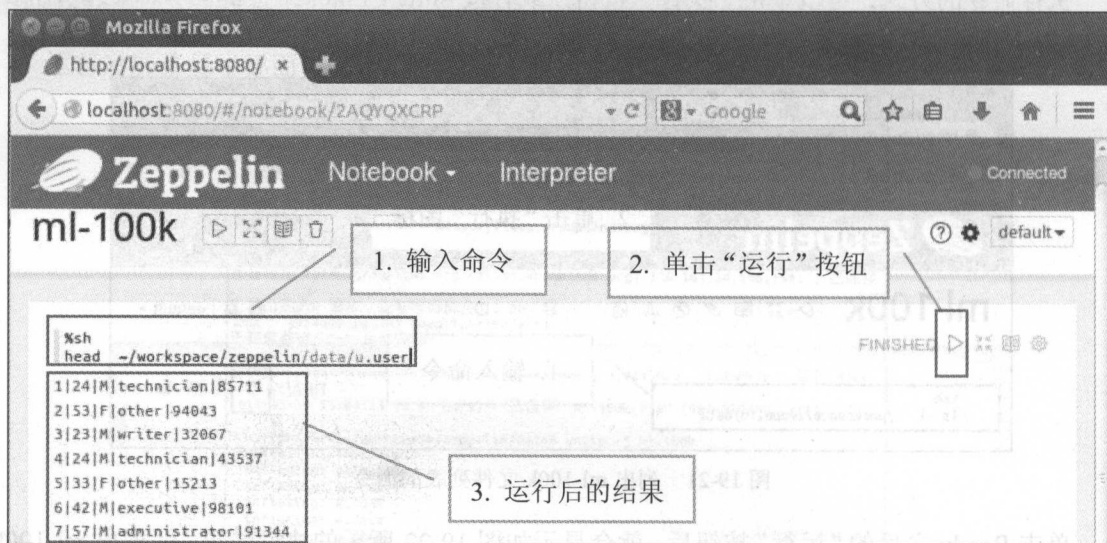


图 19-23 查看 u.user

19.6 创建临时表 UserTable

我们必须先使用 Spark 语句创建临时表 UserTable，后续才能使用 Spark SQL 进行数据分析以及数据可视化。zeppelin 默认支持 Spark 语句，所以直接输入 Spark 语句即可。你不必像上一小节一样，输入 shell 命令时还必须先输入 %sh。

步骤 01 输入 spark 程序代码

接下来，在新的段落输入 spark 程序代码。

➤ 加载用户数据文件到 userText

```
val userText= sc.textFile("file:/home/hduser/workspace/zeppelin/data/u.user")
```

➤ 创建 UserTable 类

UserTable 类的字段是：序号、年龄、职业、邮政编码。

```
case class UserTable(id:String,age:String, gender:String, occupation : String,
zipcode : String)
```


➤ 创建 userRDD

以“|”符号分隔字段读取每一个字段数据，然后使用 map 命令转换为用户数据表：UserTable(s(0),s(1), s(2), s(3), s(4))。

```
val userRDD = userText.map(s=>s.split("\\|")).map( s=>UserTable(s(0),s(1),s(2),s(3),s(4)))
```

➤ 将 userRDD 登录为临时表 UserTable

将 userRDD 登录为临时表 UserTable，后续可以使用 Spark SQL 读取这个数据表，用于数据分析以及数据可视化。

```
userRDD.toDF().registerTempTable("UserTable")
```

➤ 输出导入记录数

```
println("导入:"+ userRDD.count+"条")
```

第一次运行需要花费比较多的时间，具体过程和运行结果如图 19-24 所示。



图 19-24 创建临时表 UserTable

19.7 使用 Zeppelin 运行年龄统计 Spark SQL

上一小节中，我们已经将 userRDD 登录为临时表 UserTable，接下来就可以使用 Spark SQL，读取 UserTable。

步骤 01 输入年龄统计的 Spark SQL

输入 Spark SQL 时，必须在第一行输入%sql。%sql 主要是告诉 Zeppelin 的解释器 (Interpreter)，后续输入的命令是 Spark SQL。

在新的段落输入 Spark SQL 如下：

SPARK SQL	SPARK SQL 说明
%sql	告诉 Zeppelin 后续命令是 SQL。
select age,count(*) counts	按年龄统计记录数
from UserTable	读取 UserTable
group by age	按年龄字段分组
order by age	按年龄字段排序

具体步骤和运行的结果如图 19-25 所示。

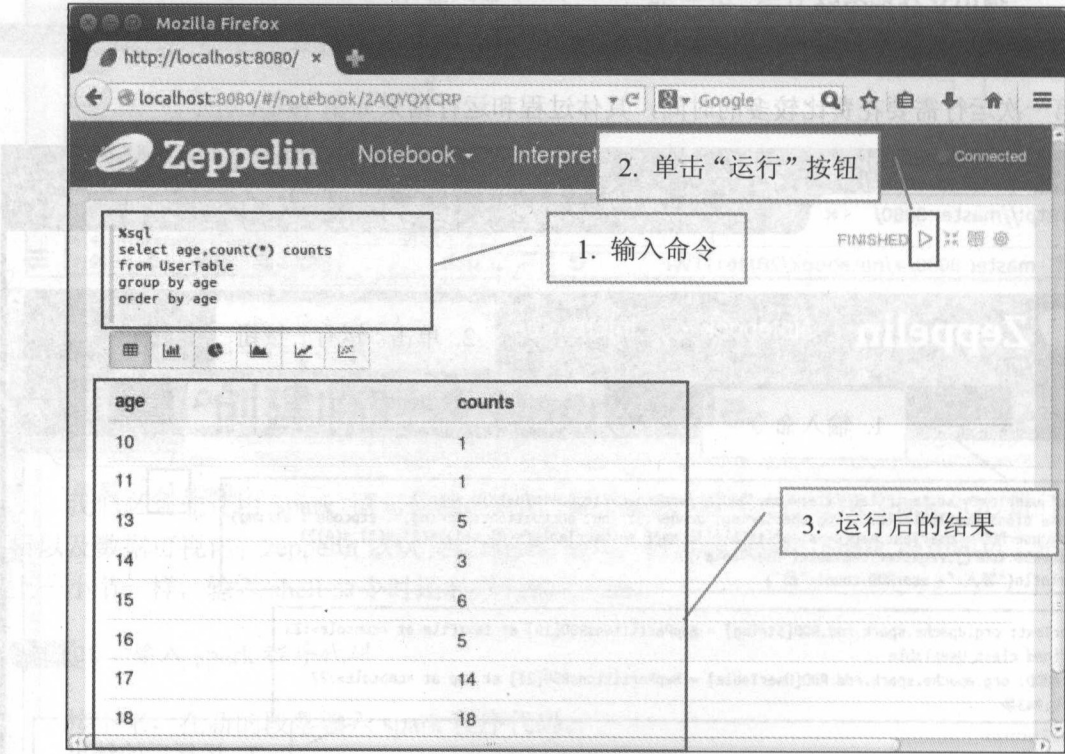


图 19-25 输入年龄统计的 Spark SQL 语句及其运行后的结果

步骤 02 显示年龄统计的柱形图

可以选择不同图标来显示不同的图形，例如单击“柱形图”图标来显示柱形图，如图 19-26 所示。

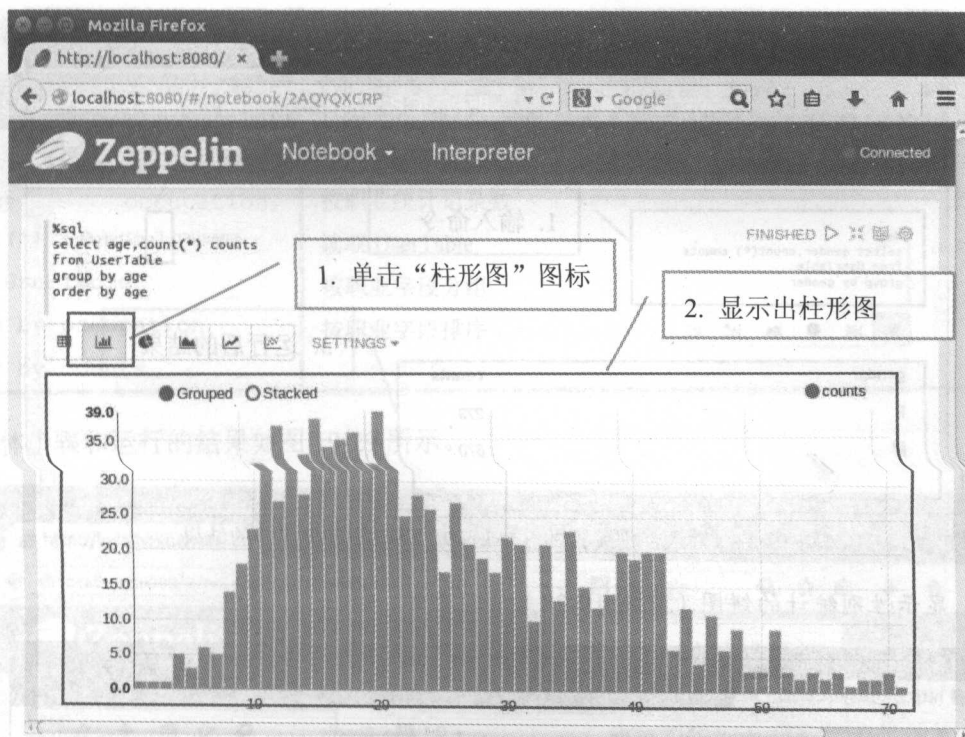


图 19-26 显示年龄统计的柱形图

19.8 使用 Zeppelin 运行性别统计 Spark SQL

步骤 01 输入性别统计的 Spark SQL

在新的段落先输入`%sql`再按 Enter 键，然后输入下列 Spark SQL：

SPARK SQL	SPARK SQL 说明
<code>%sql</code>	告诉 Zeppelin 后续命令是 SQL
<code>select gender,count(*) counts</code>	按性别统计记录数
<code>from UserTable</code>	读取 UserTable
<code>group by gender</code>	按性别字段分组

具体步骤和运行的结果如图 19-27 所示。

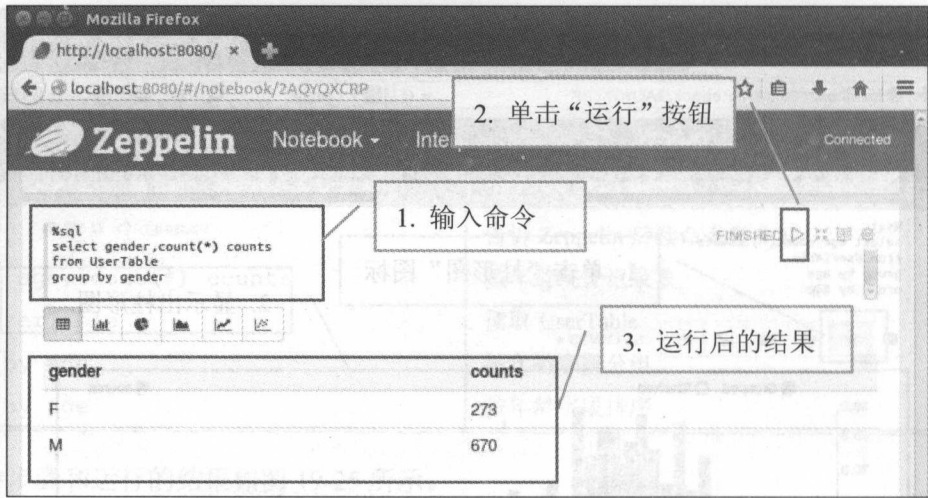


图 19-27 输入性别统计的 Spark SQL 语句及其运行后的结果

步骤 02 显示性别统计的饼图 (见图 19-28)

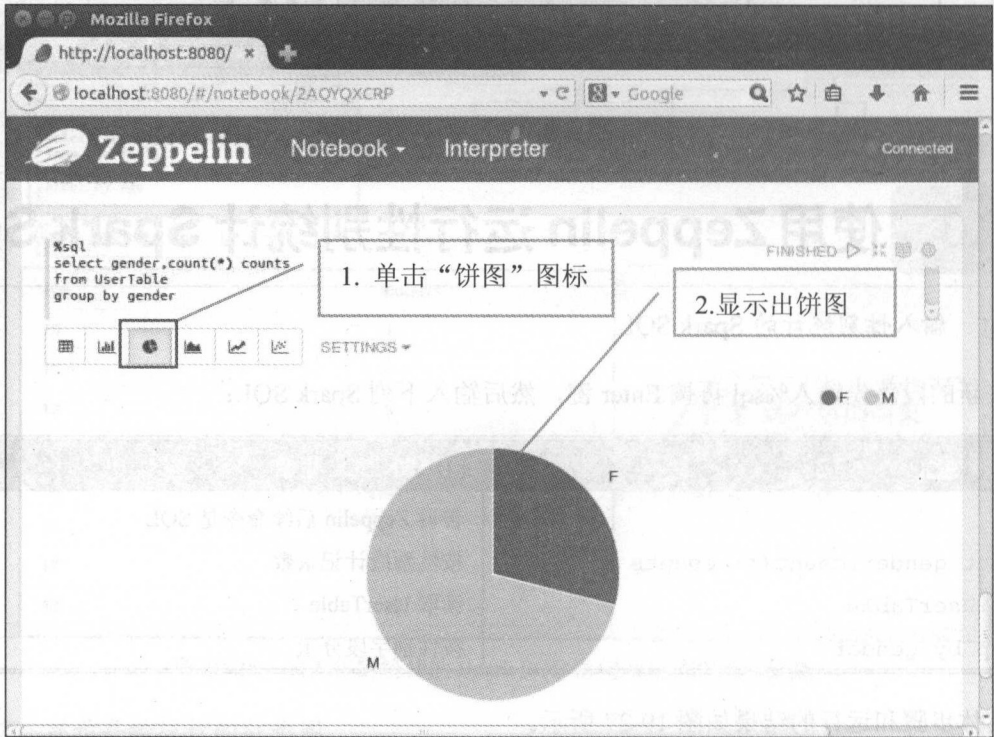


图 19-28 显示性别统计的饼图

19.9 按照职业统计

步骤 01 输入职业统计的 Spark SQL

在新的段落输入 Spark SQL 如下：

Spark SQL	Spark SQL 说明
%sql	告诉 Zeppelin 后续命令是 SQL
select occupation,	按职业统计记录数
count(*) counts	读取 UserTable
from UserTable	按职业字段分组
group by occupation	按职业字段排序
order by counts	

具体步骤和运行的结果如图 19-29 所示。

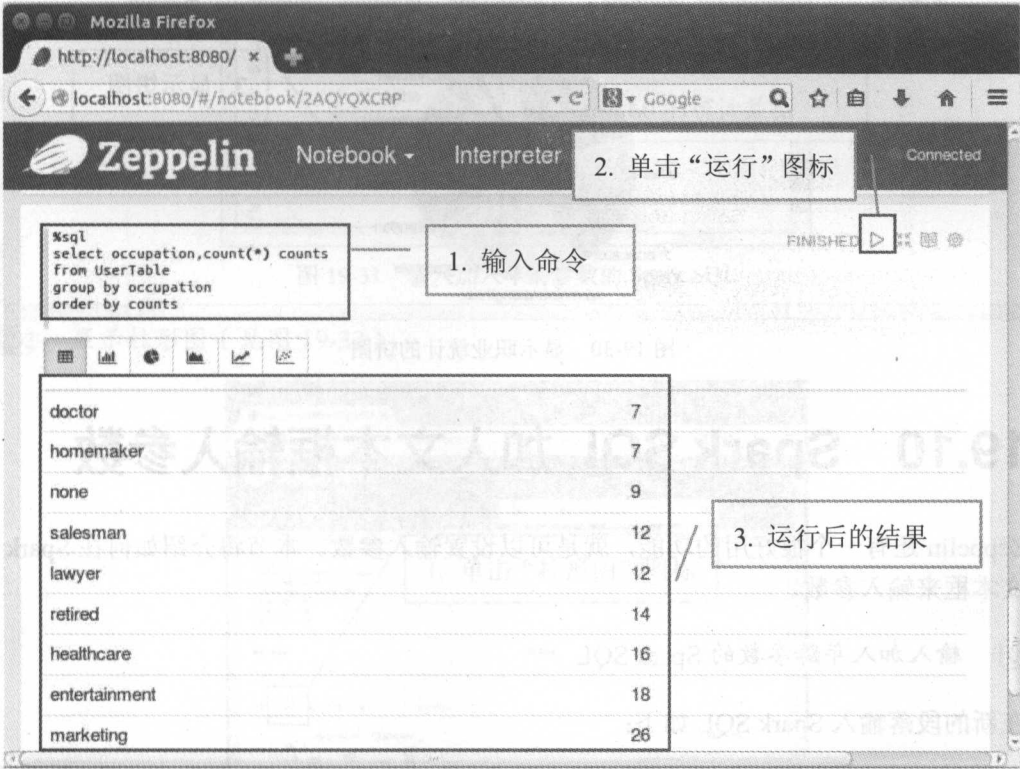


图 19-29 输入职业统计的 Spark SQL 语句及其运行后的结果

步骤 02 显示职业统计饼图（见图 19-30）

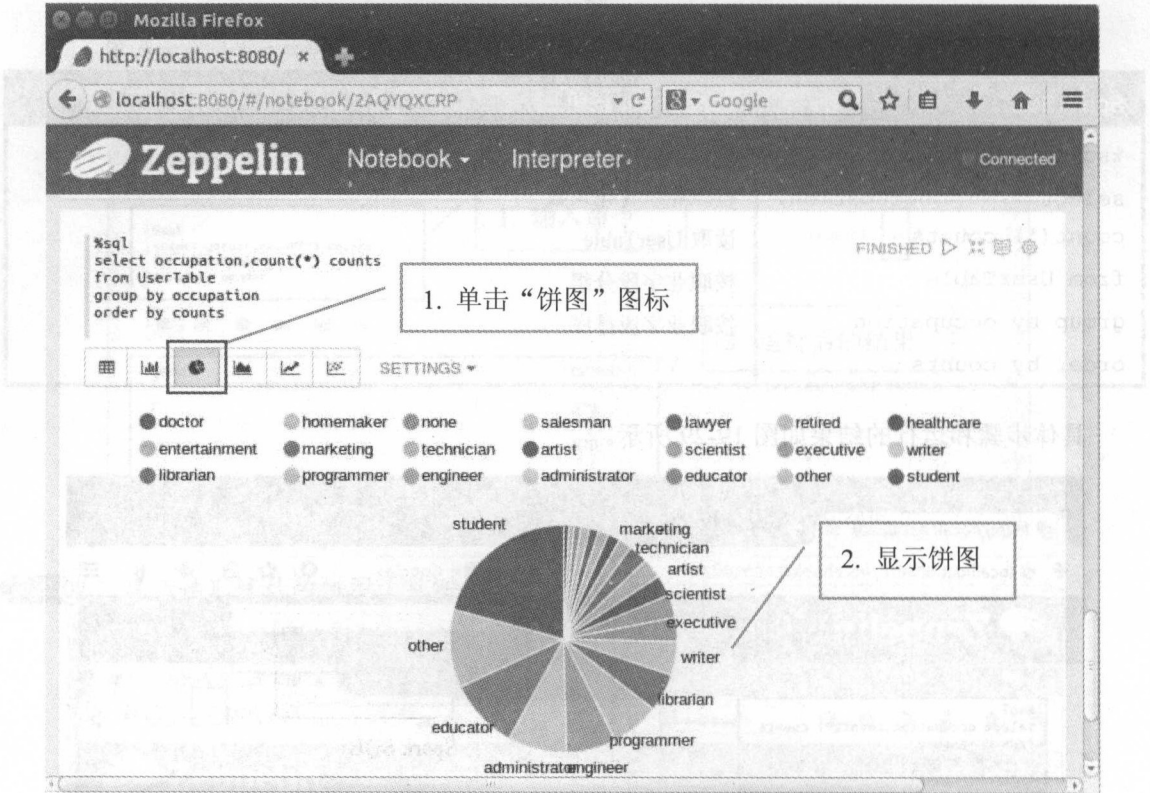


图 19-30 显示职业统计的饼图

19.10 Spark SQL 加入文本框输入参数

Zeppelin 还有一个很好用的功能，就是可以设置输入参数。本节将介绍如何在 Spark SQL 加入文本框来输入参数。

步骤 01 输入加入年龄参数的 Spark SQL

在新的段落输入 Spark SQL 如下：

Spark SQL	Spark SQL 说明
<pre>%sql select age,count(*) counts from UserTable where age>\${minAge=20} and age<\${maxAge=60} group by age order by age</pre>	<p>告诉 Zeppelin 后续命令是 SQL</p> <p>按年龄统计记录数</p> <p>读取 UserTable</p> <p>创建 minAge 文本框参数，默认值是 20</p> <p>创建 maxAge 文本框参数，默认值是 60</p> <p>按年龄字段分组</p> <p>按年龄字段排序</p>

屏幕显示界面如图 19-30 所示，运行时是以默认值 minAge=20、maxAge=60 来筛选此范围的数据。

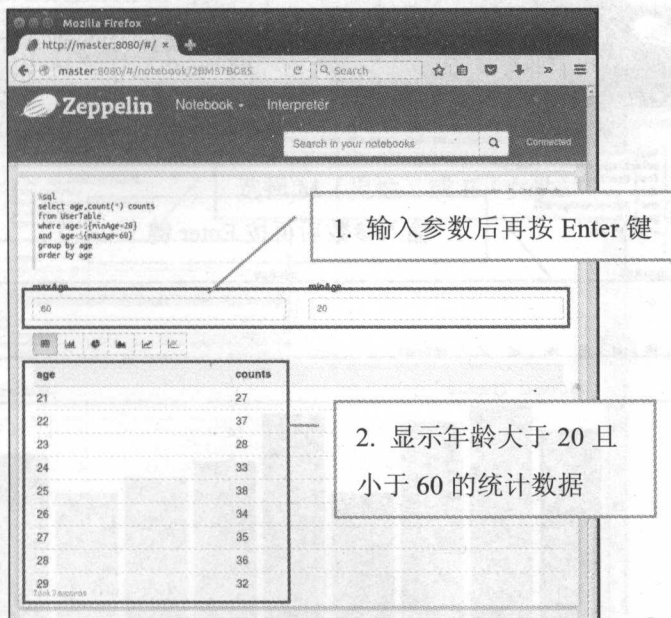


图 19-31 输入加入年龄参数的 Spark SQL

步骤 02 显示柱形图（见图 19-32）

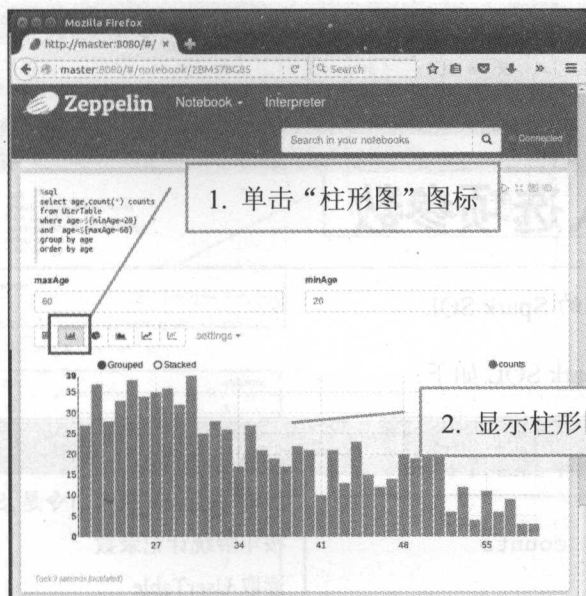


图 19-32 显示按年龄统计的柱形图

步骤 03 输入参数

我们可以输入 minAge=30、maxAge=40。随后屏幕显示如图 19-33 所示。

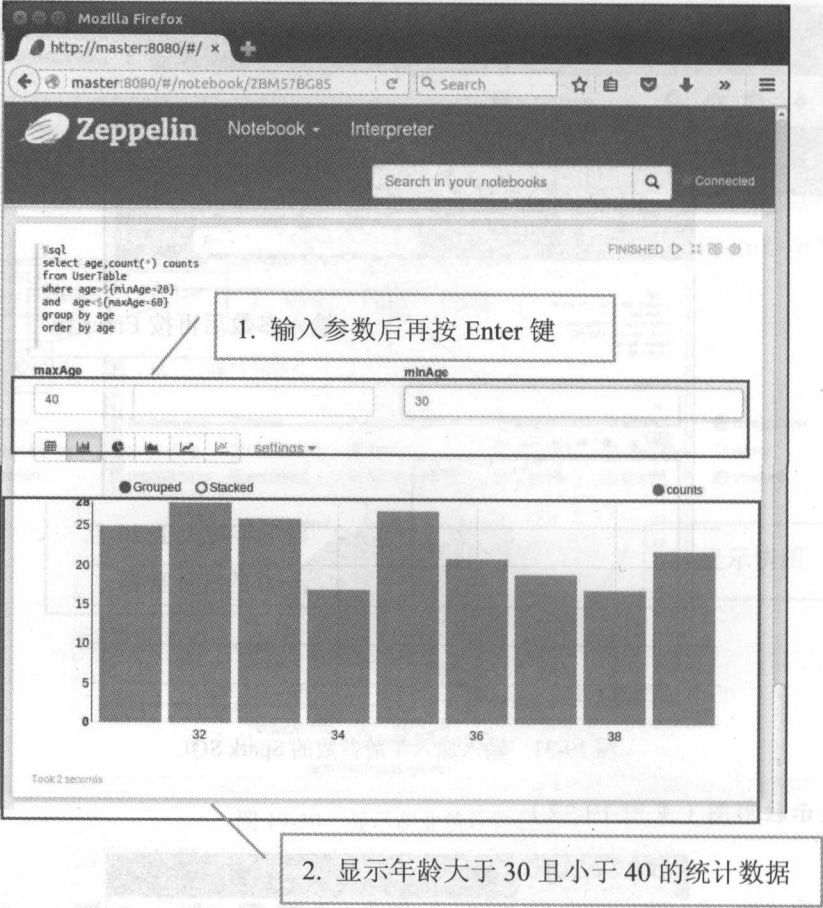


图 19-33 输入新年龄参数后的柱形图

19.11 加入选项参数

步骤 01 输入选项参数的 Spark SQL

在新的段落输入 Spark SQL 如下：

SPARK SQL	SPARK SQL 说明
<pre>%sql select age,count(*) counts from UserTable where gender="\${gender=M,M F}" group by age order by age</pre>	<p>告诉 Zeppelin 后续命令是 SQL</p> <p>按年龄统计记录数</p> <p>读取 UserTable</p> <p>创建一个 gender 选项，默认值是：M（男）；有 2 个选项 M（男）与 F（女）</p> <p>按年龄字段分组</p> <p>按年龄字段排序</p>

执行后就会出现 gender 选项的下拉菜单，如图 19-34 所示。

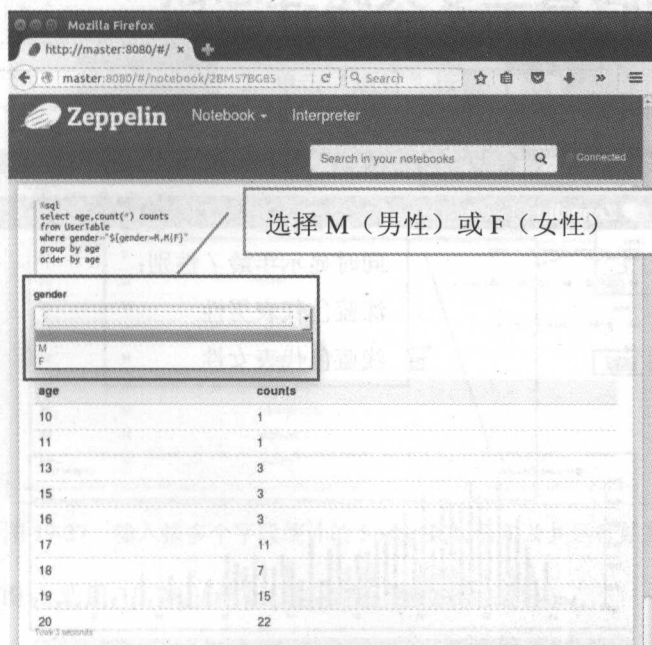


图 19-34 输入选项参数会出现下拉菜单

步骤 02 筛选性别为男性 (见图 19-35)



图 19-35 筛选性别为男性后显示男性的按年龄统计的柱形图

19.12 同时显示多个统计字段

接下来将介绍同时显示年龄 / 性别，如图 19-36 所示。

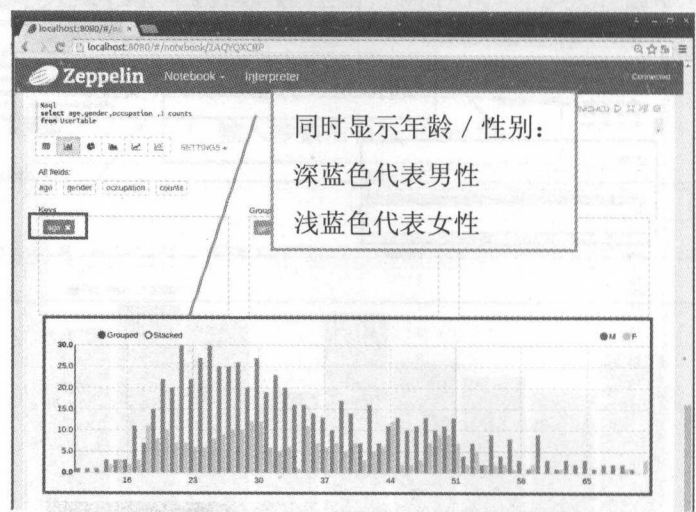


图 19-36 同时按年龄/性别统计的柱形图

步骤 01 输入 Spark SQL

在新的段落输入 Spark SQL 如下：

Spark SQL	Spark SQL 说明
<code>%sql</code>	告诉 Zeppelin 后续命令是 SQL
<code>select age,gender,occupation,</code>	显示年龄、性别、职业
<code>1 counts</code>	counts 固定为 1
<code>from UserTable</code>	读取 UserTable

counts 固定为 1，后续 Zeppelin 会自动统计。具体步骤和运行结果如图 19-37 所示。

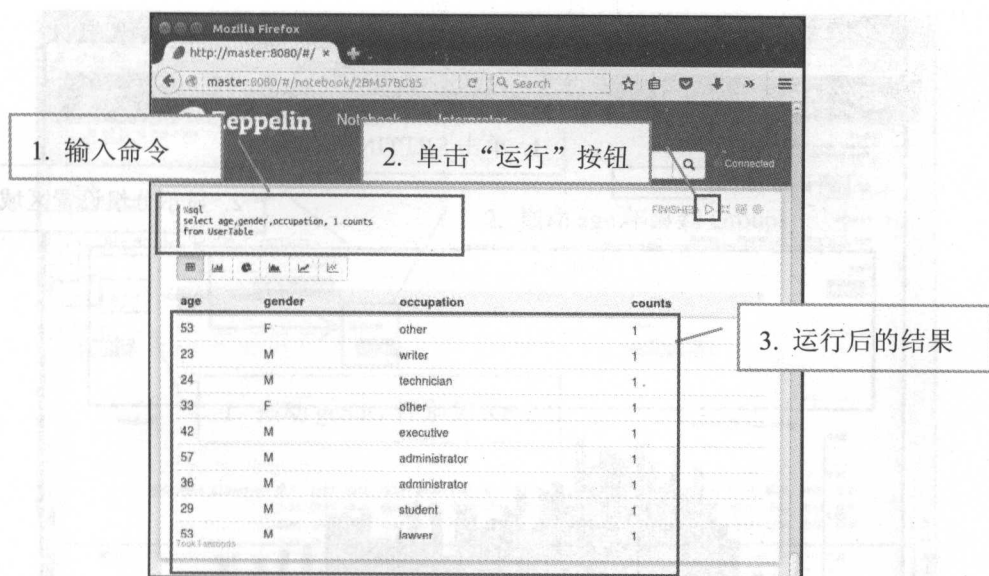


图 19-37 输入按多个字段统计的 Spark SQL 语句及其运行结果

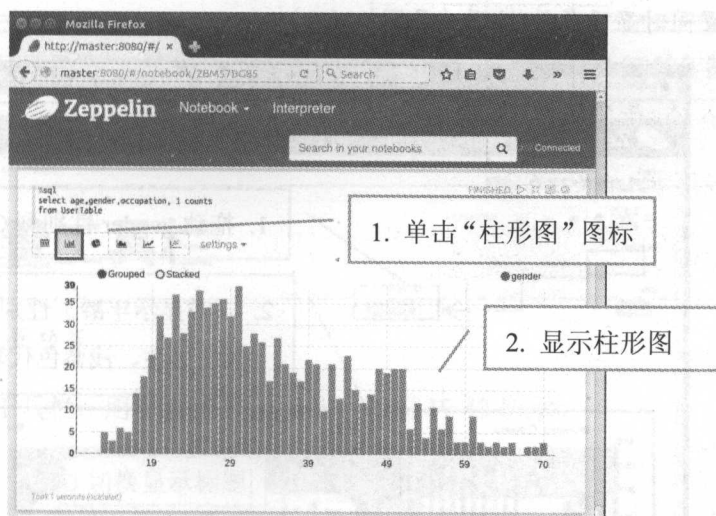
步骤 02 显示柱形图（见图 19-38）

图 19-38 显示按多个字段统计的柱形图

步骤 03 设置显示年龄/性别（见图 19-39）

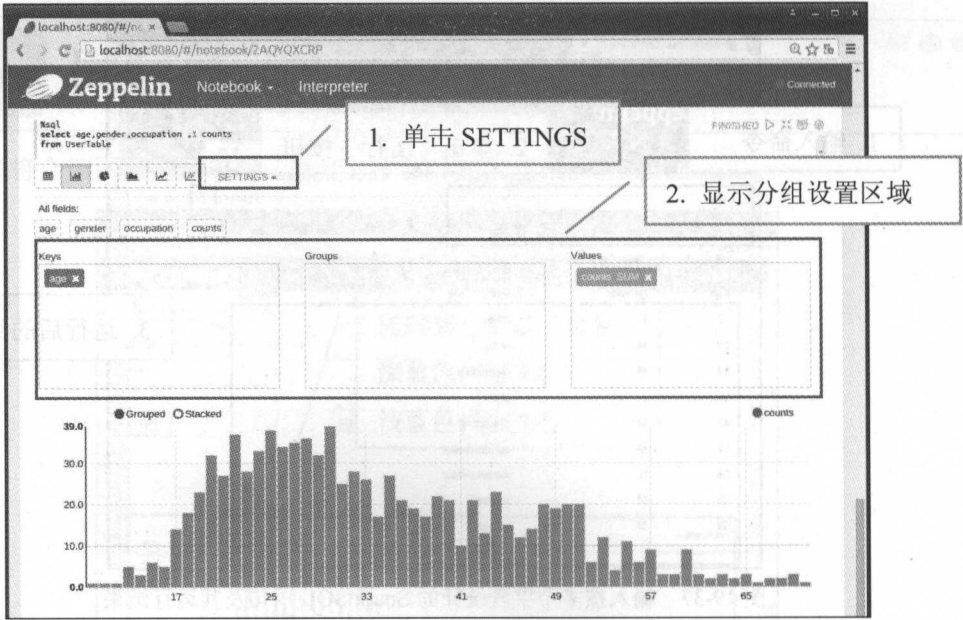


图 19-39 设置选择多个统计的字段

步骤 04 设置同时显示年龄/性别（见图 19-40）

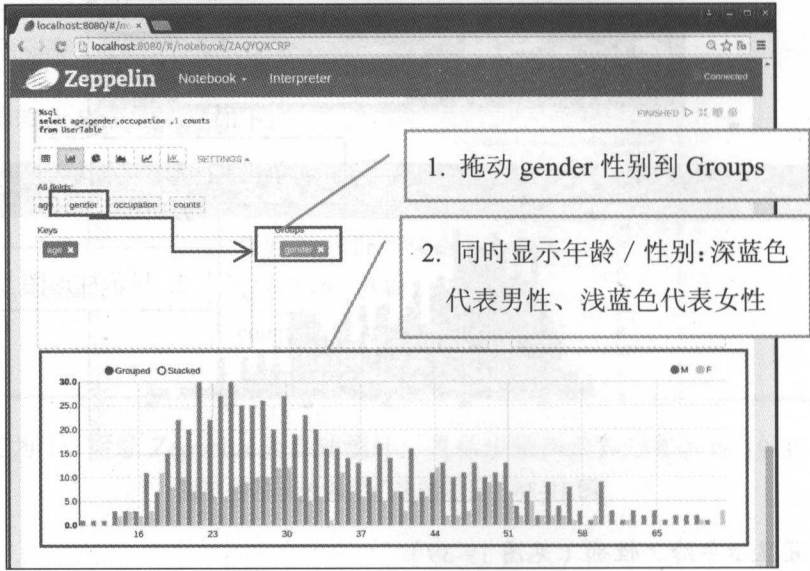


图 19-40 设置同时显示年龄/性别

步骤 05 设置显示性别 / 年龄的另外一种方式

还可以使用另一种方式显示性别 / 年龄，如下列步骤（参见图 19-41）。

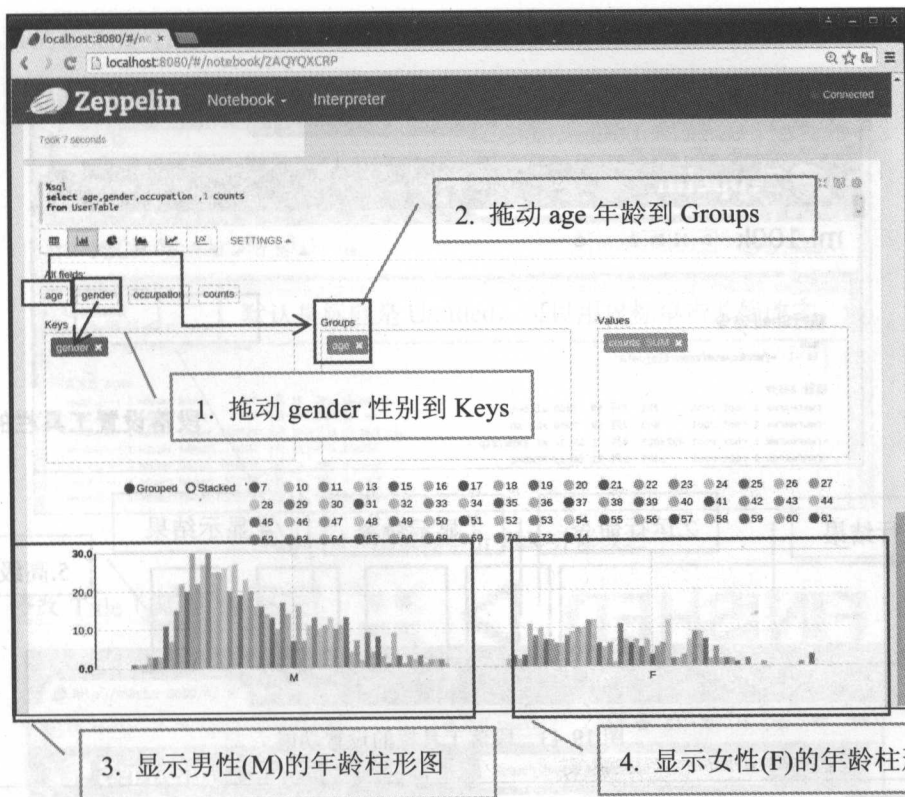


图 19-41 设置显示性别 / 年龄的另外一种方式

19.13 设置工具栏

步骤 01 工具栏的设置功能

Zeppelin 提供了工具栏，用于设置 Notebook，如图 19-42 所示。

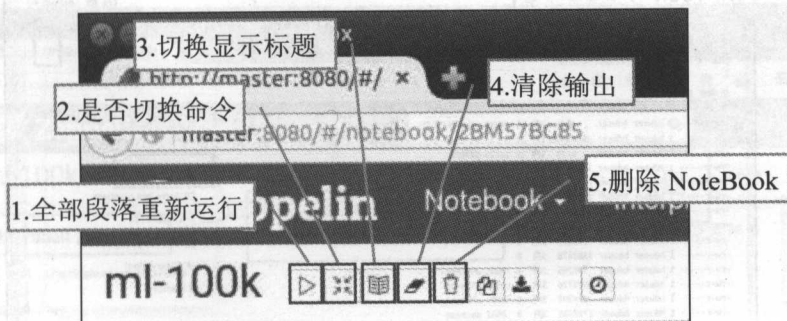


图 19-42 工具栏的设置功能

步骤 02 段落 (Paragraph) 工具栏的设置功能

在每一个段落中，Zeppelin 也提供了工具栏，用于针对段落的设置。如图 19-43 所示。



图 19-43 段落工具栏的设置功能

19.14 设置段落标题

步骤 01 设置显示 Title (见图 19-44)



图 19-44 设置显示 Title (标题)

步骤 02 用鼠标单击 Title 并修改之（见图 19-45）



图 19-45 用鼠标单击 Title 并修改之

步骤 03 修改 Title（见图 19-46）

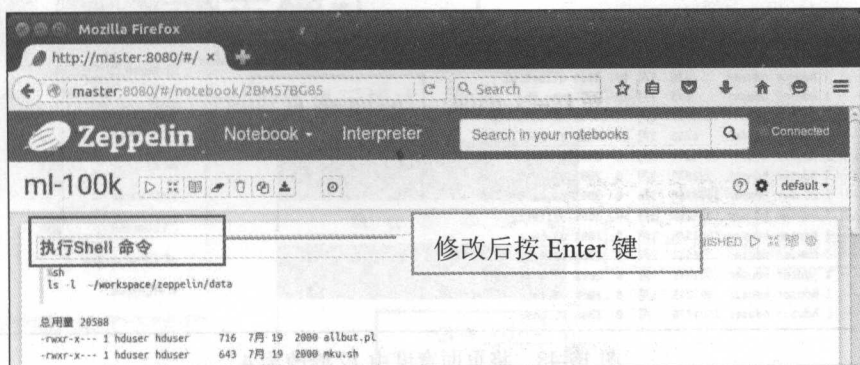


图 19-46 修改 Title

步骤 04 完成 Title 的修改（见图 19-47）

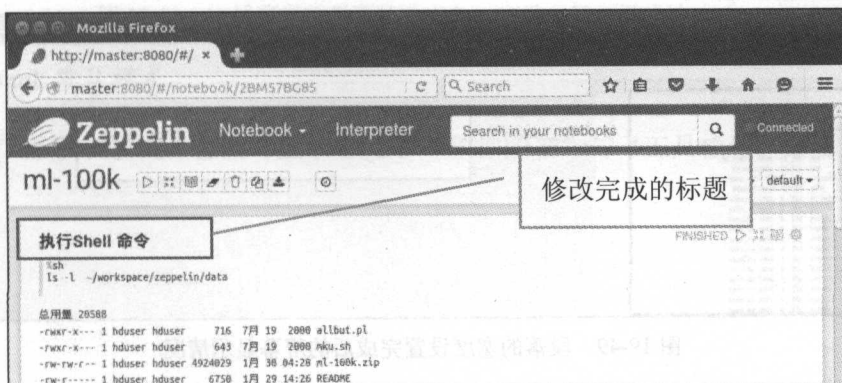


图 19-47 完成 Title 的修改

19.15 设置 Paragraph 段落的宽度

步骤 01 单击下拉菜单

段落宽度默认值是 12，也就是整个页面。可以修改为 4，则会占据页面空间 $4/12 = 1/3$ ，也就是说整体占据页面 1/3 面积。如图 19-48 所示。

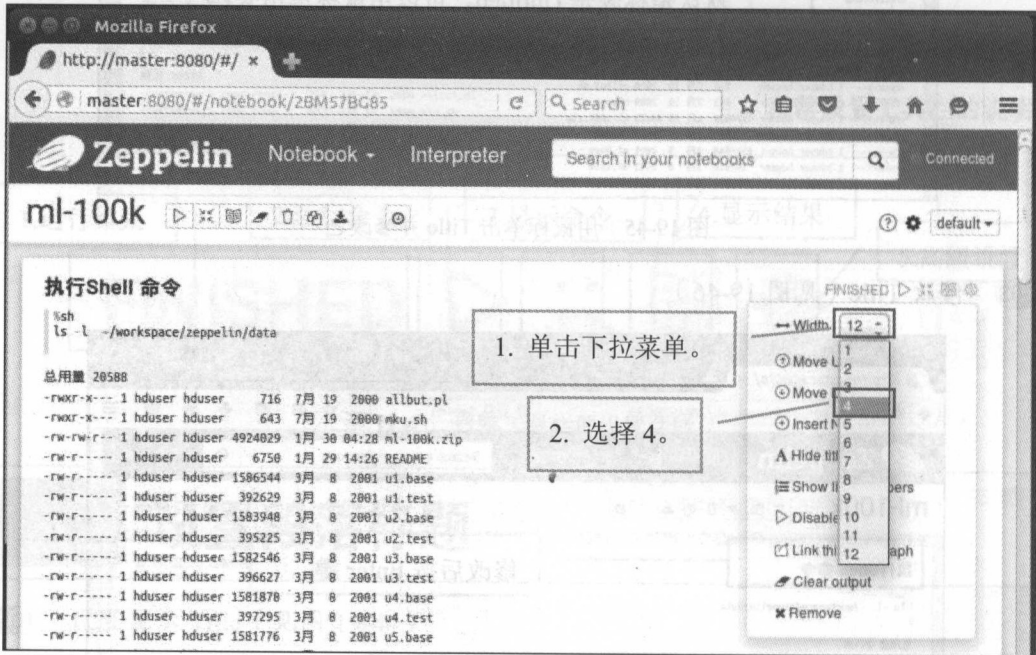


图 19-48 将页面宽度由 12 修改为 4

步骤 02 完成设置段落的宽度（见图 19-49）

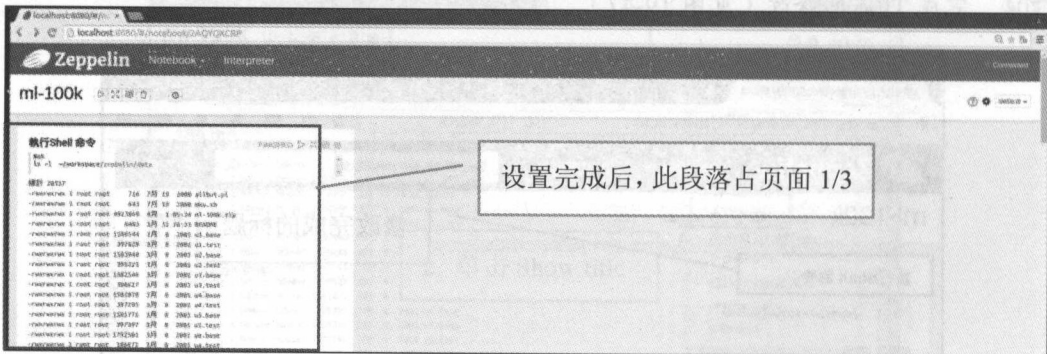


图 19-49 段落的宽度设置完成后的屏幕显示情况

步骤 03 设置段落

你可以将所有段落都设为 1/3 页面，屏幕显示界面如图 19-50 所示。



图 19-50 所有段落都设为 1/3 页面后的屏幕显示情况

19.16 设置显示模式

在界面的右上角，我们可以设置显示模式。如图 19-51 所示。

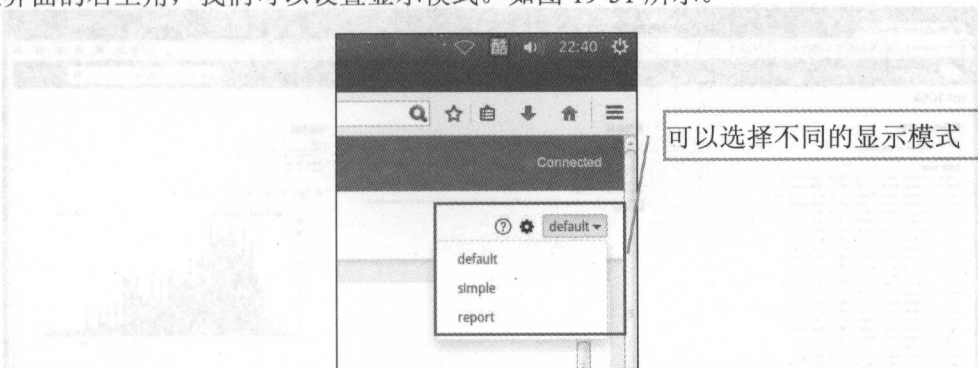


图 19-51 在屏幕显示界面的右上角，有设置显示模式的选项

步骤 01 default 默认模式

选择 default 默认模式，会同时显示 SQL、“数据图表”与“工具栏”。如图 19-52 所示。

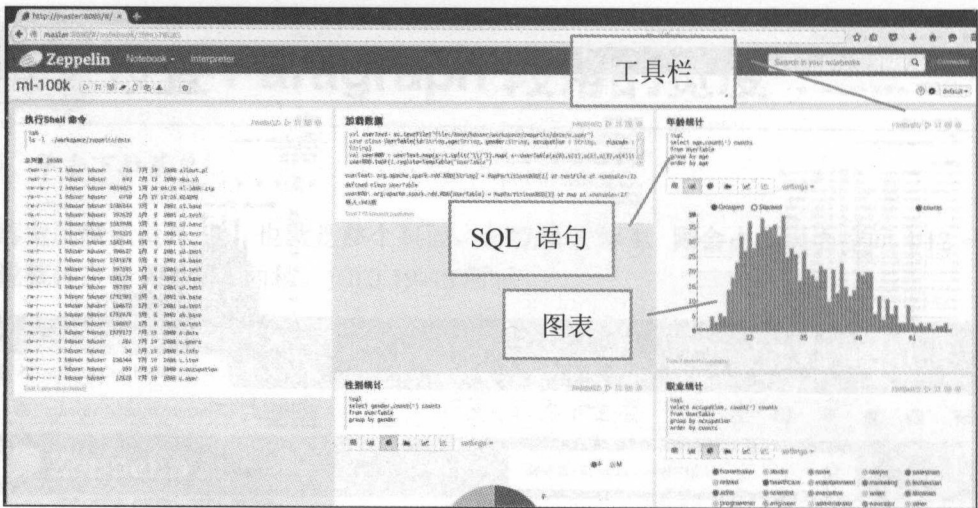


图 19-52 默认模式时屏幕会同时显示 SQL、“数据图表”与“工具栏”

步骤 02 simple 简单模式

选择 simple 简单模式会同时显示 SQL 与数据图表，但是工具栏不会显示出来，只有鼠标移至段落时才会显示出工具栏。如图 19-53 所示。

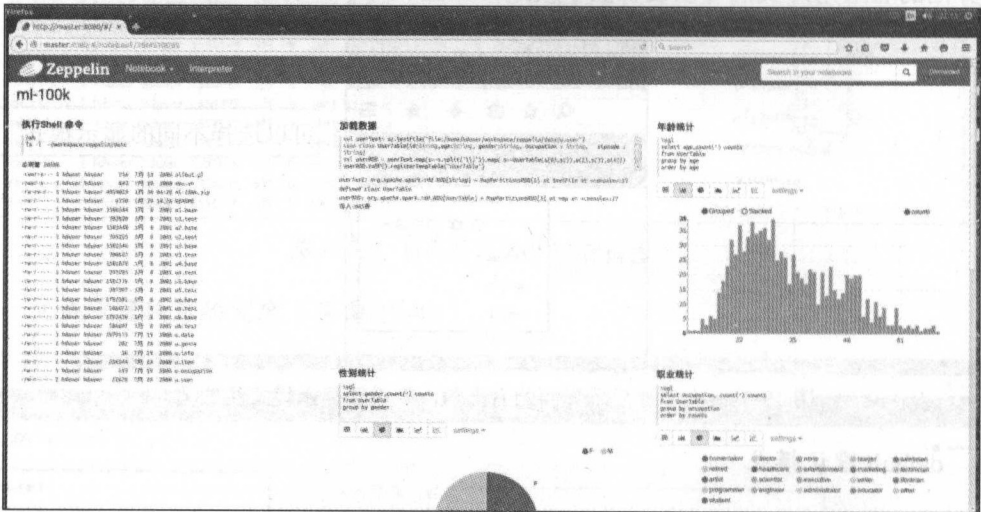


图 19-53 simple 简单模式时屏幕会同时显示 SQL 与数据图表

步骤 03 Report 报告模式

选择 Report 报告模式，那么只会显示数据图表。如图 19-54 所示。

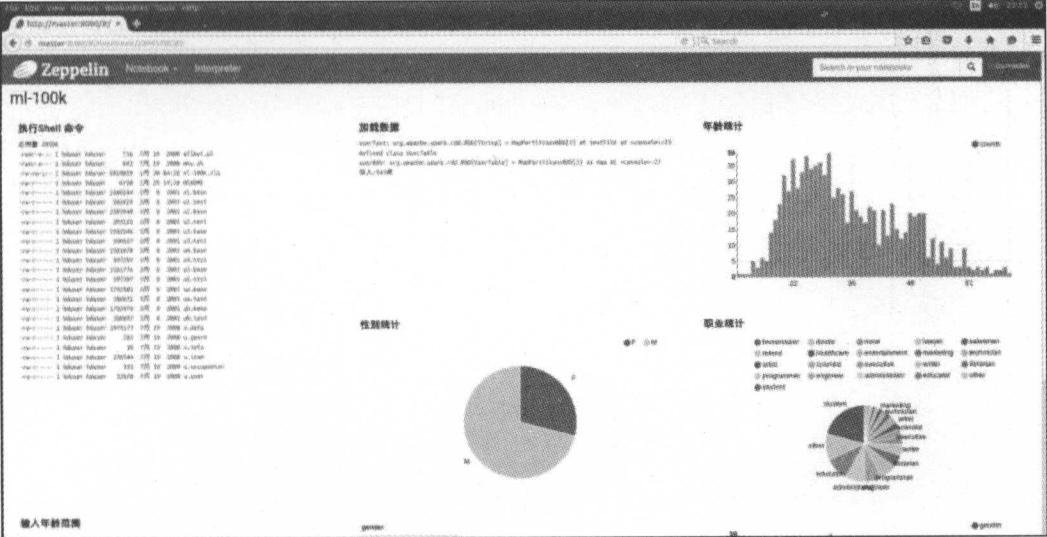


图 19-54 Report 报告模式时屏幕只会显示出数据图表

大数据分析机器学习人工智能
带来信息科技革命的第5波新浪潮

创新产业 · 大量商机 · 人才需求

一般人可能会认为大数据需要在很多台机器的环境下才能学习，实际上通过虚拟机的方法，就能在自家电脑上演练建立Hadoop集群，并且建立Spark开发环境。本书以实际操作介绍Hadoop中的MapReduce与HDFS基本概念，以及Spark中的RDD与MapReduce基本概念。

以大数据分析实际案例—MoiveLens（电影推荐引擎）、StumbleUpon（网页二元分类）、CovType（森林覆盖植被运算）、Bike Sharing（Ubike类租赁预测分析）。配合范例程序代码详解各种机器学习算法，示范如何获取数据、分析数据、建立模型、预测结果，由浅入深地介绍Spark机器学习。

清华大学出版社数字出版网站

WQBook 书文局

www.wqbook.com

ISBN 978-7-302-45375-8



9 787302 453758 >

定价：79.00元